THE UNIVERSITY
of EDINBURGH

Towards the Development of Flexible, Reliable, Reconfigurable, and High-Performance Imaging Systems

by

Jalal Khalifat

THE UNIVERSITY
of EDINBURGH

A thesis submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

The University of Edinburgh

May 2016

Declaration

I hereby declare that this thesis was composed and originated entirely by myself except where explicitly stated in the text, and that this work has not been submitted for any other degree or professional qualifications.

_____

Jalal Khalifat

May 2016

Edinburgh, U.K.

Acknowledgements

All praises are due to Allah, the most gracious, the most merciful, after three and a half years of continuous work, this work comes to end and it is the moment that I have to thank all the people who have supported me throughout my PhD study and who have contributed to the completion of this thesis.

First and foremost, I would like to express my sincere appreciation to my supervisor, Professor Tughrul Arslan, who added a lot to my research work through his superb supervision and illimitable knowledge. I deeply appreciate his kindness, patience, and continuous guidance extended to me throughout these years. In addition, I would like also to express my gratitude to my second supervisor, Dr. Alistor Hamilton, who helped me a lot in different stages of my research work.

Special thanks to my financial sponsor, the University of Bahrain (UoB) for providing me with the opportunity to conduct my research work in the UK. I would like to thank all my colleagues at the UoB for their support and encouragements during my study.

I would like to thank all members of the System Level Integration Group (SLIG) at the University of Edinburgh. In particular, my senior colleague Dr. Ali Ebrahim, who supported me so much during my study through his suggestions and guidance. I also acknowledge the contributions of Mr. Adewale Adetomi and Mr. Godwin Enemali who have been closely involved in my research work.

Last but not least, I would like to express my deepest gratitude and best wishes to all members of my family. Special thanks go to my wonderful wife and my parents for their love and support.

Lay Summary of Thesis

The strength of Field-Programmable Gate Arrays (FPGAs) lies in the reconfigurable hardware resources that allow implementation of different hardware functions on the FPGA plane by loading various binary files. Modern FPGAs consist of a large number of these resources that allow building System on Chips (SoCs). Unlike software solutions, FPGA functions are composed using Hardware Description Language (HDL) and then synthesised to produce the function's binary file. This method is more complex than the software designing flow, but with the evolution of FPGAs, automated high-level synthesis tools have been developed to accelerate and simplify this process. In addition to the flexibility of FPGA devices, hardware functions on FPGAs can be executed in parallel to achieve a higher performance compared to the sequential software solutions. A Dynamic Partial Reconfiguration (DPR) feature enhances the flexibility further by allowing the functionality of part of the system to be changed at run-time while the other parts are functioning normally. For the aforementioned reasons, FPGAs are becoming a suitable platform for many applications in different fields. This thesis presents the use of FPGAs in image processing and discusses the advantages of FPGAs over current solutions such as Application Specific Integrated Circuits (ASICs) and Graphics Processing Units (GPUs). These advantages of FPGAs have been obtained by implementing a number of imaging algorithms for different stages of the Image Processing Pipeline (IPP) on FPGA devices. Moreover, it presents a novel reliable imaging system that uses DPR to swap tasks over time. This practice increases system performance, reliability, and flexibility and reduces the area utilisation and power consumption. The system can be easily integrated into reconfigurable operating systems due to its unbounded flexibility resulting from the use of FPGA

Abstract

Current FPGAs can implement large systems because of the high density of reconfigurable logic resources in a single chip. FPGAs are comprehensive devices that combine flexibility and high performance in the same platform compared to other platform such as General-Purpose Processors (GPPs) and Application Specific Integrated Circuits (ASICs). The flexibility of modern FPGAs is further enhanced by introducing Dynamic Partial Reconfiguration (DPR) feature, which allows for changing the functionality of part of the system while other parts are functioning. FPGAs became an important platform for digital image processing applications because of the aforementioned features. They can fulfil the need of efficient and flexible platforms that execute imaging tasks efficiently as well as the reliably with low power, high performance and high flexibility. The use of FPGAs as accelerators for image processing outperforms most of the current solutions. Current FPGA solutions can to load part of the imaging application that needs high computational power on dedicated reconfigurable hardware accelerators while other parts are working on the traditional solution to increase the system performance. Moreover, the use of the DPR feature enhances the flexibility of image processing further by swapping accelerators in and out at run-time. The use of fault mitigation techniques in FPGAs enables imaging applications to operate in harsh environments following the fact that FPGAs are sensitive to radiation and extreme conditions.

The aim of this thesis is to present a platform for efficient implementations of imaging tasks. The research uses FPGAs as the key component of this platform and uses the concept of DPR to increase the performance, flexibility, to reduce the power dissipation and to expand the cycle of possible imaging applications. In this context, it proposes the use of FPGAs to accelerate the Image Processing Pipeline (IPP) stages, the core part of most imaging devices. The thesis has a number of novel concepts. The first novel concept is the use of FPGA hardware environment and DPR feature to increase the parallelism and achieve high flexibility. The concept also increases the performance and reduces the power consumption and area utilisation. Based on this concept, the following implementations are presented in this thesis: An

implementation of Adams Hamilton Demosaicing algorithm for camera colour interpolation, which exploits the FPGA parallelism to outperform other equivalents. In addition, an implementation of Automatic White Balance (AWB), another IPP stage that employs DPR feature to prove the mentioned novelty aspects. Another novel concept in this thesis is presented in chapter 6, which uses DPR feature to develop a novel flexible imaging system that requires less logic and can be implemented in small FPGAs. The system can be employed as a template for any imaging application with no limitation. Moreover, discussed in this thesis is a novel reliable version of the imaging system that adopts novel techniques including scrubbing, Built-In Self Test (BIST), and Triple Modular Redundancy (TMR) to detect and correct errors using the Internal Configuration Access Port (ICAP) primitive. These techniques exploit the datapath-based nature of the implemented imaging system to improve the system's overall reliability. The thesis presents a proposal for integrating the imaging system with the Robust Reliable Reconfigurable Real-Time Heterogeneous Operating System (R4THOS) to get the best out of the system. The proposal shows the suitability of the proposed DPR imaging system to be used as part of the core system of autonomous cars because of its unbounded flexibility. These novel works are presented in a number of publications as shown in section 1.3 later in this thesis.

.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

APU	Application Processing Unit
ASIC	Application Specific Integrated Circuit
ASSP	Application-Specific Standard Product
AWB	Automatic White Balance
BRAM	Block Random Access Memory
CCD	Charge-Coupled Device
CFA	Colour Filter Array
CGRA	Coarse Grained Reconfigurable Architecture
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
CMT	Clock Management Tile
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRISP	Coarse-Grained Reconfigurable Instruction Set Processor
DDR	Double Data Rate
DIP	Digital Image Processing
DLP	Data Level Parallelism
DMA	Direct Memory Access
DMR	Dual-Module Redundancy
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processor
EAC	Empty Area Compaction
ECC	Error Correction Code
EDF	Earliest Deadline First
FAEDF	Finishing Aware EDF
FIFO	First In First Out
FIT	Failure in Time
FMC	FPGA Mezzanine Card
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FU	Function Unit
GPP	General Puprose Processor
GPU	Graphical Processing Unit
GUI	Graphical User Interface
GWA	Gray World Assumption
HCE	Hot Carrier Effect
HD	High-Definition
HDL	Hardware Description Language
HLS	High Level Synthesis
HP	High Performance
HPC	High Performance Computing
HW µK	Hardware Microkernel

ICAP	Internal Configuration Access Port
ICM	Internal Configuration Manager
ICs	Instruction Cells
IDE	Integrated Design Environment
IOP	Input Output Peripheral
IP	Intellectual Property
IPL	Instruction Level Parallelism
IPP	Image Processing Pipeline
ISE	Integrated Synthesis Environment
JPEG	Joint Photographic Experts Group
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MBU	Multiple Bit Upset
MFW	Multiple Frame Write
MMCM	Mixed Mode Clock Manager
MORA	Multimedia Oriented Reconfigurable Array
MPMC	Multi Port Memory Controller
MSE	Mean Square Error
MTBFs	Mean Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time to Repair
NCD	Native Circuit Description
ND	Negative Diagonal
NGC	Native Generic Circuit
NGD	Native Generic Database
NoC	Network-on-Chip
NPI	Native Port Interface
NRE	Non-Recurring Engineering
OpenCL	Open Computing Language
PAR	Place and Route
PCAP	Processor Configuration Access Port
PD	Positive Diagonal
PL	Programmable Logic
PLL	Phase-Locked Loop
PRA	Perfect Reflector Assumption
PS	Processing System
PSNR	Peak Signal to Noise Ratio
QIS	Quartus Integrated Synthesis
R4THOS	Robust Reliable Reconfigurable Real-Time Heterogeneous Operating System
RAM	Random Access Memory
RGB	Red Green Blue
RGBE	Red Green Blue Emerald
RGGB	Red-Green-Green-Blue
RICA	Reconfigurable Instruction Cell Array
RISC	Reduced Instruction Set Computing

RM	Reconfigurable Module
RP	Reconfigurable Partition
RR	Reconfigurable Region
RTL	Register-Transfer-Level
SAD	Sum of Absolute Difference
SDK	Software Development Kit
SEL	Single-Event Latchup
SEM	Soft Error Mitigation
SET	Single-Event Transient
SEU	Single-Event Upset
SoC	System-on-Chip
SRAM	Static Random Access Memory
SRL	Shift Register Lut
SSD	Sum of Squared Difference
SW µK	Software Microkernel
TMR	Triple-Module Redundancy
UCF	User Constraints file
VDMA	Video Direct Memory Access
VHSIC	Very High Speed Integrated Circuit
VTC	Video Timing Controller
XDC	Xilinx Design Constraints file
XPS	Xilinx Platform Studio
XST	Xilinx Synthesis Technology

Chapter 1 : Introduction

The evolution of the electronics industry has allowed the technology to be involved deeply in our lifestyle and to change all aspects of our lives. The high demand in the technology market drove the growth of the industry to be one of the fastest growing industries. In this context, the image-processing field is one of fastest evolving technology fields. It refers to both digital and analog image processing. It is the process of performing operations, called algorithms, on images to enhance the images or extract useful information from the image plane. Specifically, the purpose of image processing is divided into five groups [1]:

- Visualisation: observe poorly visible objects.
- Image sharpening and restoration: create better images.
- Image retrieval: seek an image of interest.
- Measurement of pattern: measure objects in an image.
- Image recognition: Distinguish an object in an image.

This field has become quite important due to the large number of applications that need to process scene representations in all life aspects including medical, biological, security and surveillance, robotic, biometric, remote sensing, and photo and video applications such as cameras. These applications require various processing architectures to accommodate different processing requirements. Moreover, because of the large amount of data in a single image, image-processing algorithms need a high computational power to achieve their goals. Throughout the history of image processing, many processing solutions have been used to process multimedia data based on the requirements of the available technologies. Each solution has its own strengths and weaknesses. Traditionally, General Purpose Processors (GPPs) were used to process data based on the stored-programme computing model in which the

function instructions are stored in the memory and then executed sequentially according to the fetch decode execute cycle. This platform solution is very flexible and allows tasks to be fetched at any time and in any order. However, the GPP operational clock frequency limits the performance due to the sequential nature of the technology. According to Moor's law [2] and Dennard scaling [3], the operational frequency cannot be increased further due to the thermal and power dissipation. Another solution is to use multiprocessors to increase the system parallelism; this solution has some limitations in the ability of the application to support parallel tasks.

Over time, image-processing applications began requiring high-throughput and real-time processing capabilities, which cannot be achieved by GPPs. Therefore, a new set of processing platforms is used in the field: ASICs, Digital Signal Processors (DSPs), Graphical Processing Units (GPUs), and FPGAs. ASICs are generally employed in most of the modern camera systems because of their high performance and power efficiency. Conversely, their functionality cannot be changed once they are fabricated; this limits the overall number of applications that can be deployed. Figure 1.1 shows a comparison between these different platforms with respect to performance, productivity, programmability, and power efficiency.

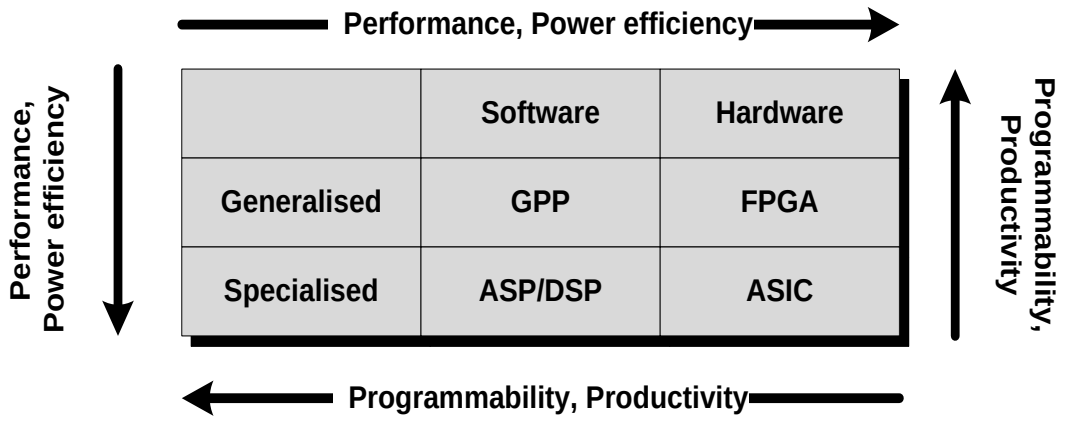


Figure 1.1 Comparison between computing platforms in terms of performance, power efficiency, programmability, and productivity

As shown in the figure, ASICs can achieve better performance and power efficiency, but they are lacking flexibility and productivity. The programmability term is primarily related to software-based platforms because of their processing style, but in the case of FPGAs, the programmability can be achieved by reconfiguring the FPGA components as needed to be a software-like platform [4]. The programmability of FPGAs cannot be compared to GPPs, but it gives them a significant advantage over ASICs in addition to the time-to-market aspect. Moreover, modern FPGAs have a performance and power efficiency close to ASICs due to the new interconnect architecture that connects components and to the deployment of new transistor technology, which dissipates less power. In this context, FPGAs lead the reconfigurable computing to bridge the gap between the high performance provided by ASICs and the flexibility provided by GPPs.

The desired platform for image processing should combine all these aspects in the same platform—this combination can be found in modern FPGA computing platforms. FPGAs consist of reconfigurable logic blocks, which can be constantly reconfigured to form new functions after manufacturing. Modern FPGAs have become highly dense devices containing many special components. Some of these FPGAs contain more than 20 billion transistors on the chip [5]. These devices can be used to build complex system on the same chip. Another important factor when choosing the platform for specific applications is time-to-market. Traditionally, FPGA functions were implemented using Hardware Description Language (HDL), which describes the Register-Transfer-Level (RTL). Because this method is time consuming, the need of newer and more efficient methods has become important. Today, the leading FPGA companies have increased the overall productivity and shortened development time by shifting the development process toward a higher level of abstraction. This has been achieved by employing high-level synthesis tools that use software programme languages instead of describing the RTL manually. These tools convert the high-level languages into RTL descriptions efficiently, with much less time. Each company has introduced its own set of tools. This helps the field to be developed faster and to be more attractive.

The Flexibility of FPGAs is the major advantage over ASIC platforms. Traditionally, the flexibility is limited to implementing different designs on the same plane. In modern FPGAs, introducing DPR feature extends the flexibility to a higher level, which minimises the gap with software platforms and gives FPGAs the software look and feel. DPR allows part of the design to be modified at run-time while the remaining parts of the system are operating. This feature enables efficient use of resources by allowing tasks to share the same logic in a time-multiplexed style and thus reducing the overall resource utilisation. In this context, less resource utilisation means less power consumption. Furthermore, DPR can reduce power consumption further by swapping out the idle tasks at run-time. This way of deploying hardware tasks in the FPGA is similar to deploying software tasks using an operating system. Moreover, DPR can enhance the performance of the system by allowing larger functions to share the plane of the FPGA, which enables them to increase their amount of parallelism. Introducing DPR feature enhances the flexibility, performance, power consumption, and productivity of FPGAs one step forward, which is another advantage over other platforms.

The use of FPGAs in image processing field instead of ASICs extends the flexibility and the productivity to a higher level and keeps the performance and power consumption relatively similar. This allows for in-field repairs, modification, and upgrades, which means that FPGAs can support a wide range of algorithms and standards. Moreover, it protects image processing applications from obsolescence by allowing them to be upgraded to the latest technologies dynamically. Conversely, FPGAs are sensitive to high level of radiations because of the used Static Random Access Memory (SRAM) technology. The reconfigurability of FPGAs can overcome this problem in a self-healing style through continuous repairs of the internal components internally without any external circuitry. This method allows FPGA-based image-processing applications to operate with no problems in harsh environments such as high radiation areas and in space. The main aim of this thesis is to investigate the suitability of FPGA adoption for developing image-processing algorithms. This thesis looks deeply into exploiting the FPGA flexibility to develop

highly flexible imaging systems that can be used for various image processing applications.

1.1 Thesis Objectives

The main objective of this thesis is to propose and develop an FPGA-based imaging architecture that exploits the available features of modern FPGAs to establish a generic imaging platform. Precisely, it aims to exploit the DPR feature and the FPGA parallelism ability to develop flexible, high-performance, and reliable imaging systems. These aspects are explained in detail in the following paragraphs.

❖ Flexible

The reconfigurability of the FPGA adds significant flexibility to the developed platform. This flexibility can be divided into three aspects:

First, it enables the platform to develop any imaging application by customising the sub-functions and the relationship between them. This action allows for building any imaging-based functionality.

Second, it enables the platform to update, modify, and change algorithms and standards for a specific functionality, which keeps the system up-to-date and allows the platform to work easily under different conditions.

Third, it enables the platform to be integrated easily into bigger environments by using the universal standards and centralised control panel.

This thesis aims to achieve these aspects by applying DPR to a specific region on the FPGA where all these functions can be customised dynamically.

❖ High Performance

FPGAs are high-performance devices due to their hardware nature. This thesis aims to enhance the performance further by exploiting the reconfigurable resources at run-time. In this way, the functions share the FPGA resources over time, which allows for increasing the overall parallelism. This thesis aims to investigate applying the aforementioned techniques to image processing designs to enhance their overall performance. However, these techniques need to use DPR feature in which the functions are reconfigured dynamically through a specific port. This sequential

process can limit the overall performance. This thesis also aims to use some configuration techniques that enable the configuration process to achieve higher throughput.

- **Reliable**

Most commercial FPGAs are based on SRAM technology, which is sensitive to high levels of radiation. This radiation can cause faults in the developed design. To develop a robust imaging application that can work anywhere, the platform should adopt fault detection and correction mechanisms. This thesis aims to develop and use self-healing and self-test techniques to increase the system availability through the efficient use of the internal configuration port. Moreover, it aims to decrease the overall correction time compared to other available techniques.

1.2 Novelty and Contribution

First, this thesis presents the design and architecture of a novel DPR imaging system that enables and simplifies the implementation of imaging applications. The novel architecture allows for implementing designs with low resource utilisation, low power consumption, high scalability, and high performance. In this architecture, pre-fetching and variable task size techniques are used for enhancing the performance and flexibility. The architecture is ready to be integrated into reconfigurable operating systems to be used in an efficient way, as presented in this thesis.

To enable the system to operate in harsh environments, a novel version of the imaging system is presented. The version can detect and correct errors using different schemes including scrubbing, redundancy, and self-test techniques. These schemes exploit the data-path based nature of the imaging system to enhance the overall reliability by reducing the down time and the recovery time.

Two accelerators for IPP stages are implemented to investigate the aforementioned aspects. Adams-Hamilton demosaicing engine for colour interpolation is implemented using two approaches: high-level synthesis [6] and RTL. The high-level synthesis uses high-level languages such as C or C++ to implements the parts of

design, while RTL approach describes the design part using HDL such as VHDL. An evaluation study for the two approaches is conducted to investigate the best method for integrating image processing cores on FPGAs. Moreover, the DPR AWB engine is presented using the two approaches. This implementation investigates the use of DPR to reduce the power consumption and resource utilisation, and to enhance the performance. The results show that the DPR implementation is much better than the static implementation in all aspects

The work presented in this thesis is one of the projects carried out by the System Level Integration Group (SLIG) at the University of Edinburgh. The thesis refers to a number of concepts that are the contribution of other researchers in SLIG such as R3TOS and R4THOS Kernels, allocation and scheduling algorithms, R3TOS Internal Configuration Manager (ICM), and multiple task configurations.

1.3 Publications

A number of publications have emerged from this work as follows:

Conferences

1. **An Efficient Implementation of the Adams-Hamilton's Demosaicing Algorithm in FPGAs**
   Khalifat. J, Ebrahim. A, Arslan. T
   In 10th International Symposium on Reconfigurable Computing: Architectures, Tools, and Applications (ARC 2014), pp. 205-212, 2014

2. **A Novel Dynamic Partial Recon guration Design for Automatic White Balance**
   Khalifat. J, Arslan. T
   The NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 9-14, 2014

3. **A dynamic partial reconfiguration design for camera systems**
   Khalifat. J, Ebrahim. A, Adetomi. A, Arslan. T
   The NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 1-7, 2015

4. **A Platform for Secure IP Integration in Xilinx Virtex FPGAs**
   Ebrahim. A, Benkrid. K, **Khalifat. J**, Hong. C
   The International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1-6, 2013

1.4 Thesis Outline

This thesis consists of seven chapters. The remaining chapters address the following topics.

Chapter 2: Introduction to FPGAs and Dynamic Partial Reconfiguration

This chapter has two parts; the first part introduces fundamental features of FPGAs in terms of the architecture, design flow, and main components. It presents different design flows for FPGA devices. The DPR feature and its design flow are introduced in this part. The DPR feature deployments for high performance, power reduction, area saving, and reliability are also introduced with the relevant research works. The second part introduces the fundamentals of image processing systems. It presents solutions to fulfil the requirements of real-time image processing. It addresses the basic IPP in detail. Moreover, it discusses the available computing platforms for image-processing and investigates the best platforms for the field. Based on the literature, the chapter presents a number of image processor architectures and discusses their features.

Chapter 3: Efficient Implementation of the Adams-Hamilton Demosaicing Algorithm on FPGAs

This chapter presents the design and architecture for an efficient implementation of the Adams-Hamilton interpolation algorithm that constructs the full image pixels. The architecture exploits the FPGAs parallelism to increase the performance of the algorithm to meet the processing requirements of modern cameras. This chapter discusses two approaches for implementing the algorithm. In addition, it performs an evaluation and comparison study to determine the best use of each approach.

Chapter 4: Dynamic Partial Reconfiguration Implementation for Automatic White Balance on FPGA

This chapter presents a novel implementation of one of the AWB algorithms that processes the images to keep the colour of objects constant under different illumination conditions. The architecture exploits the parallelism and DPR features to

increase the performance of the algorithm, reduce resource utilisation, and decrease power consumption to meet the processing requirements of modern cameras. A comparison between the static and DPR designs is presented to show the effect of using DPR on performance, resource utilisation, and power consumption. Also presented is the implementation of a configuration engine that enhances the configuration process of swapping tasks.

Chapter 5: A Dynamic Partial Reconfiguration Design for Camera Systems on FPGA

This chapter presents an architecture and design for a DPR imaging system that provides significant flexibility for developing imaging applications. It shows the features of this architecture and its capabilities, as well as the effect of its use on the architecture to improve power consumption, performance, area utilisation, and flexibility. Moreover, the chapter shows an enhanced version of the architecture to enhance the performance and flexibility. Finally, it presents a proposal for integrating the system with a reconfigurable operating system to increase the system efficiency and to show a real system deployment.

Chapter 6: Reliable Dynamic Partial Reconfiguration for Imaging Systems

This chapter presents a reliable version of the implemented imaging system, which adopts many techniques such as scrubbing, self-test, and redundancy to enable the system to operate in harsh environments. The chapter details the integration of these techniques with the system. Moreover, it evaluates the system reliability under upsets by injecting errors at various blocks and locations in the FPGA configuration memory. It presents a novel built-in self-test technique, which detects most upsets within a specific region with low resource overhead and less recovery time. Finally, the chapter presents a redundancy technique on the most important part of the system to reduce the overall system resource overhead.

Chapter 7: Conclusion and Future Work

This chapter concludes the research presented in the thesis and addresses the remaining open issues and future work.

Chapter 2 : Introduction to FPGAs, Dynamic Partial Reconfiguration, and Image Processing Systems

2.1 Introduction to FPGAs and DPR

In the history of digital systems, many generations have been revealed since the invention of the first transistor in 1947 [7]. The capacity of integrated transistors has increased over the years to produce powerful systems. Today, the number of transistors in commercial integrated circuits has reached over 5.5 billion for commercial Central Processing Unit (CPU) and 20 billion in FPGAs [8]. The factors of speed, flexibility or programmability, and power consumption vary for the different types of digital systems such as the CPU, GPU, DSP, ASIC and FPGA. CPUs have the most flexible architecture, but the performance is not suitable for applications with high processing demands. Conversely, ASICs have a high-speed architecture that is customised for a particular application and cannot be changed after fabrication. Unlike the aforementioned types, FPGAs are a high-performance type of digital system that can be configured after manufacturing to create different functionalities. The flexibility they have gives them the advantage over ASICs while maintaining the high performance, which is already an advantage over CPUs. The combination of high performance and flexibility in one environment enables FPGAs to meet the demands needed by wide range of applications (see Figure 2.1).

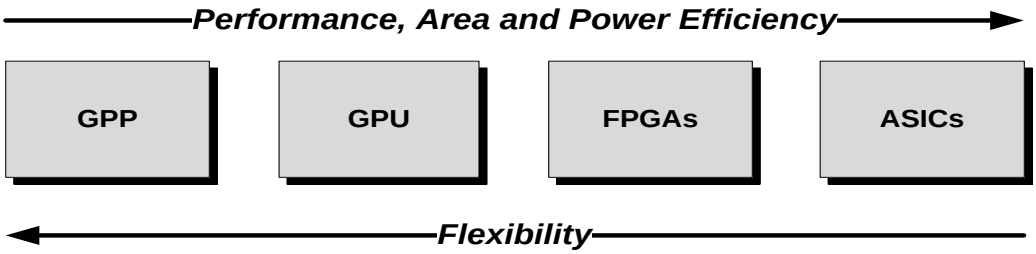


Figure 2.1 Computing platforms, performance vs. flexibility

FPGAs are integrated circuits that can be reconfigured to change their functionality. An FPGA consists of an array of reconfigurable blocks as a fixed layer. These blocks can be connected together using a hierarchy of reconfigurable interconnects on top of the fixed layer. The manner of connecting these blocks forms a specific functionality [9]. The reconfigurable interconnects consist of configuration memories that store the information that defines the fixed layer and its functionality. The first FPGA, the XC2064, was invented in 1985 [10]; this device had a small array of reconfigurable blocks. Since that time, the number of reconfigurable blocks has increased dramatically to reach millions of reconfigurable blocks in modern FPGAs. Moreover, modern FPGAs contain special reconfigurable resources such as DSPs and memory blocks to deal efficiently with complex problems such as floating-point operations. Modern FPGAs have become a real competitor to ASICs. They can be used to implement System-on-Chips (SoCs), which contain components such as processors, memories, and controllers, as in the case of the Zynq-7000 all-programmable SoC [11]. This technology opens the door for the industry to find an environment other than ASICs that could enable high-performance applications with low development cost and short time-to-market, as all components are pre-fabricated and, as mentioned, with high flexibility [12]. The flexibility primarily occurs through the reconfigurability feature due to the nature of FPGAs. This feature makes it possible to upgrade the FPGA-based systems at any point to increase their lifetime as much as possible. The manufacturers go further with FPGAs. They add the ability to change the functionality of a subsystem or part of it online without changing or stopping the adjacent running functions. This feature is called DPR. This feature, which enhances the design performance, resources utilisation, and power consumption, will be addressed later in this chapter.

Since the invention of the first FPGA, FPGAs have become an essential part of the digital system market. Over the years, manufacturers of FPGAs have tried to close the gap with other competitors such as ASICs and GPUs. They added many features to the FPGAs to be suitable for hundreds of applications ranging from simple ones to the most complex applications, which need high and intensive processing power. According to Grand View Research [13], the global FPGA market is expected to

reach $9.882 billion by 2020 (compared to $6 billion in 2015) [14]. The main applications of FPGAs, based on this report, are in the automotive industry, as most companies have already begun addressing such applications. In addition, data processing, military, aerospace, and telecommunication applications will be part of the expected FPGA market. The FPGA market is divided between a few companies that dominate the entire market. Xilinx and Altera are leading the market by 90% of the revenues [15], and the remaining is divided between Lattice Semiconductor, Microsemi, and QuikLogic companies.

This part prepares the reader who is not familiar with FPGA technology to better understand the material presented in this thesis. It presents the architecture and the capabilities of currently available FPGAs: specifically, the Zynq all-programmable SoCs. It describes the tools and methods to implement FPGA-based systems. Moreover, it details the DPR feature and its flow. It also discusses the reliability issues in these devices and reviews related works.

2.1.1 FPGAs and Design Flow

Traditionally, designers used gate level notations as a way of implementing designs on early FPGAs. Gate level notations are the use of logic gates only such as AND, OR, NOT, and BUF in the design. With the evolution of digital design, the RTL has been used to describe digital circuits [16]. RTL is used in Hardware Description Languages (HDL) such as Very High Speed Integrated Circuit (VHSIC) HDL (VHDL) and Verilog. RTL primarily consists of two parts; the first part is the combinational logic, which performs the functions of the circuit. The second part is the register, which synchronises the functions of the circuit with the clock. The HDL looks like C language in most of its syntax, but it is not sequential and has some notations that support the concurrency feature of the hardware to give space for parallelism. In HDL, the designer has full control of all aspects of the design such as signals and ports. The code at this level is translated to lower levels using special synthesis tools. One disadvantage of the HDL is that it is time consuming for the designers. Because of that, FPGA companies tried to add a new level of abstraction at the top of the digital design (see Figure 2.2) by using programming languages to

develop digital circuits [17]. A few high-level languages have been used for this purpose include SystemC, Handel-C, and SA-C [18]. Recently, the two key companies in the field have introduced their own ways to enhance the productivity and shorten the development time. Xilinx have introduced the High Level Synthesis (HLS) tool based on C, C++, SystemC, or Open Computing Language (OpenCL) languages [19]. Altera have introduced a single design environment for heterogeneous systems called OpenCL, which translates the OpenCL kernels into hardware accelerators. [20].

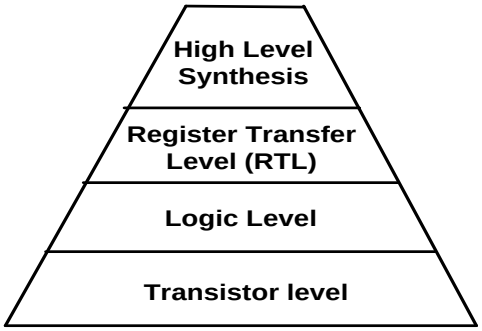


Figure 2.2 Levels of digital design

2.1.1.1 FPGA Design Flow

Despite the differences between the vendors of FPGAs, they follow a similar design flow regardless of the tools and the types of file extensions they use in their tools. The design flow consists of several stages to create the final bitstream, which has the configuration data to be loaded in a particular FPGA. Each FPGA chip has a different bitstream format. The tools should have all the information about each chip and the format of the final bitstream. In general, the design flow is comprised of the following steps: synthesis, implementation, and programming-bitstream generation.

Synthesis: Generally in this stage, an RTL specified design file is transformed into a netlist file that describes the gate-level components of the implemented design using specialised synthesis tool [21]. The output file should be ready to be processed by the next stage. If a high level of abstraction is used, an extra step is required to be performed by the high-level synthesis tool. After the high-level language is converted into RTL representation using a special tool, the RTL is converted into a

gate-level description. The extension of the stage output file varies depending on the vendor tool and the generation. Table 2.1 shows the file extension after each stage for different tools and generations. In Xilinx Integrated Synthesis Environment (ISE) Design Suite, the HDL is converted to a binary Native Generic Circuit (NGC) netlist file using Xilinx Synthesis Technology (XST). In Vivado Design Suite, all stages are represented using one file format with different content. The HDL is converted to design check point (DCP) file format using Vivado Synthesis (See Figure 2.3). In Altera, the Quartus Integrated Synthesis (QIS) tool is used to convert the HDL file to gate-level description (See Figure 2.4).

Implementation: In this stage, the process is divided into sub-stages; first, the input files are translated and merged into one design file, then the gate level descriptions are mapped to the device resources, and finally the mapped resources are placed and routed into the FPGA. These steps are processed based on the available design and timing constraint information. The aforementioned steps are the general steps for the implementation stage among all vendors' tools, but they differ in the way or in the name of that step. In ISE Design Suite, the steps are Translate, MAP, and Place and Route (PAR). In translate sub-stage, all NGC files are merged in a single Native Generic Database (NGD), which contains a detailed description of the hardware resources required by the design in the target device. In Map sub-stage, the tool maps all the logic presented in NGD file to resources of the target device by generating a Native Circuit Description (NCD) file. In PAR sub-stage, the tool generates the final NCD file containing the ready placed and routed design for generating bitstream. The constrained information for the design is kept in User Constraints file (UCF). In Vivado Design Suite, the implementation is divided into three sub-stages: opt_design, place_design, and route_design. Opt_design optimises the logic of the design to make it easier to fit onto the target device [22]. Place_design places the design onto the available resources of the target device while route_design routes and connects all the component of the design based on the target device resources. All sub-stages are represented using the same DCP file with different content. Vivado tool allows the use of non-project mode, which does not need any physical files to save the output of each stage. This mode uses a volatile memory to keep the

processed data in each stage. The constrained information for the design is kept in Xilinx Design Constraints file (XDC). In Altera, the QIS is used for implementation stage. All processed information are kept in the database (See Figure 2.3 and Figure 2.4).

Bitstream generation: In this stage, the placed and routed information are converted into a binary file representing the configuration data of the design. The binary file—bitstream—is generated using a bitstream generator tool. In ISE and Vivado Design Suites, Bitgen and write_bistream are used to generate bitstreams. On the other hand, Altera uses its own assembler to generate the bitstream.

Table 2.1 FPGA design flow in ISE, Vivado, and Altera Tools

Software	Xilinx ISE	File extension	Vivado	File extension	Altera Quartus II	File extension
Synthesis	XST	.ngc	Vivado Synthesis	.dcp	Quartus II integrated synthsis (QIS)	Saved in database
Implementation	Trasnlate	.ngd	Opt_design	.dcp	Quartus II integrated synthsis Fitter (QIS)	
Implementation	MAP	.ncd	Place_design	.dcp		
Implementation	PAR	.ncd	Route_design	.dcp		
BitGen	Bitgen	.bit	Write_bitstream	.bit	Assembler	.Pof or .Sof

The design flow for Xilinx ISE and Vivado is shown in Figure 2.3 and for Altera is presented in Figure 2.4. Table 2.1 shows the differences between various tools and the file extensions allocated to each stage. High-level synthesis is required when the entry data is in the format of high-level languages such as C, C++, or systemC. For example, the Vivado HLS tool has the following design flow [6]: 1) Compile, execute, and debug the high-level language code; 2) Synthesise the code into RTL implementation; 3) Verify the RTL; 4) Package the RTL into an IP format (see Figure 2.5).

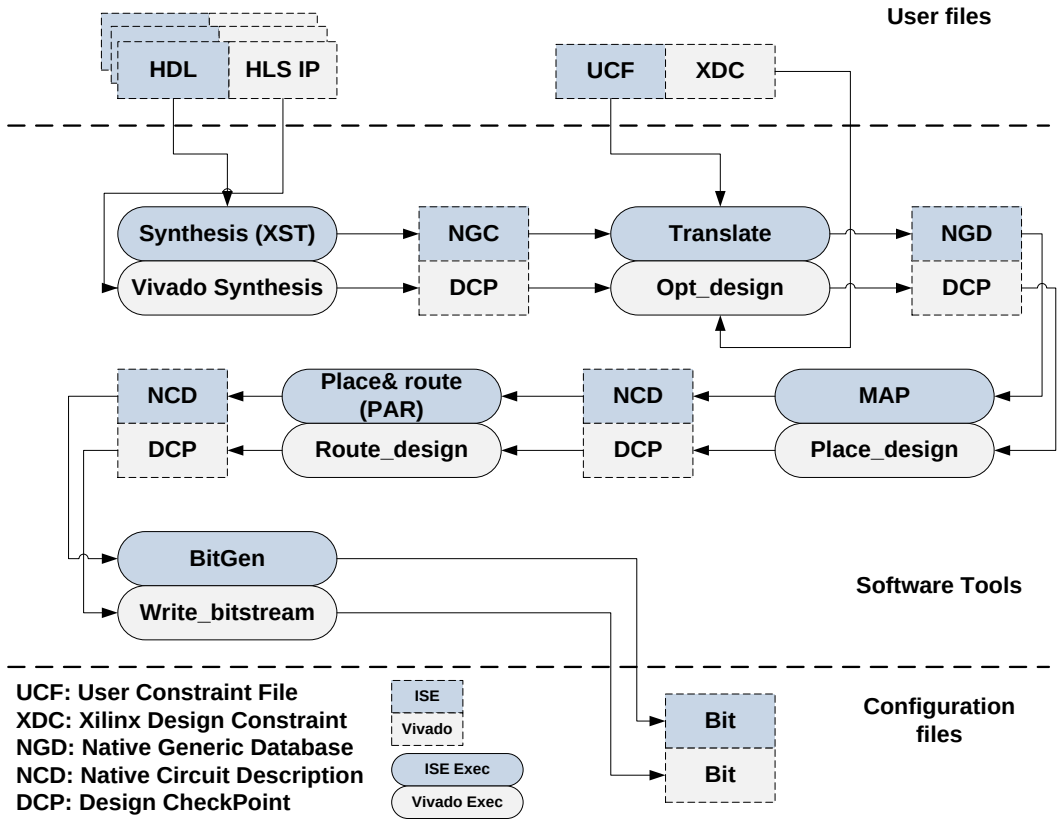


Figure 2.3 ISE and Vivado design flow

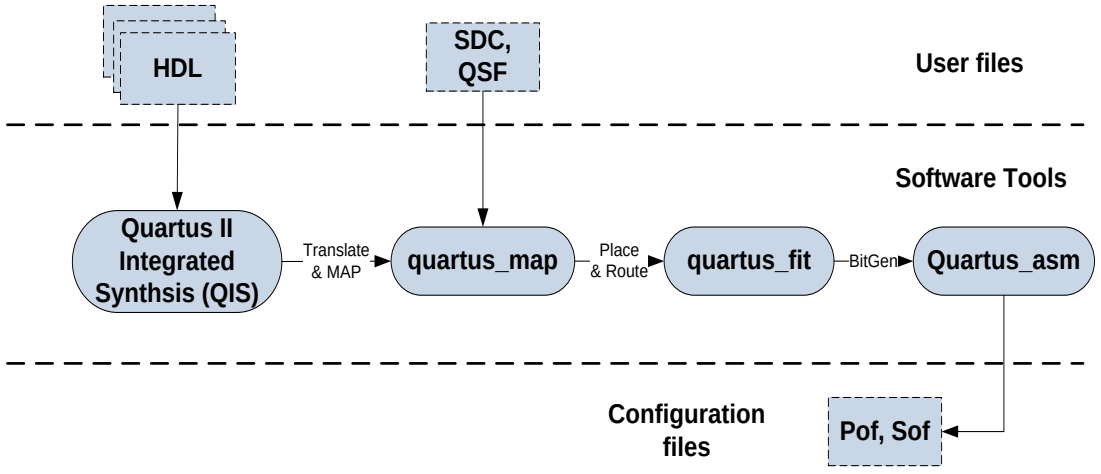


Figure 2.4 Altera design flow

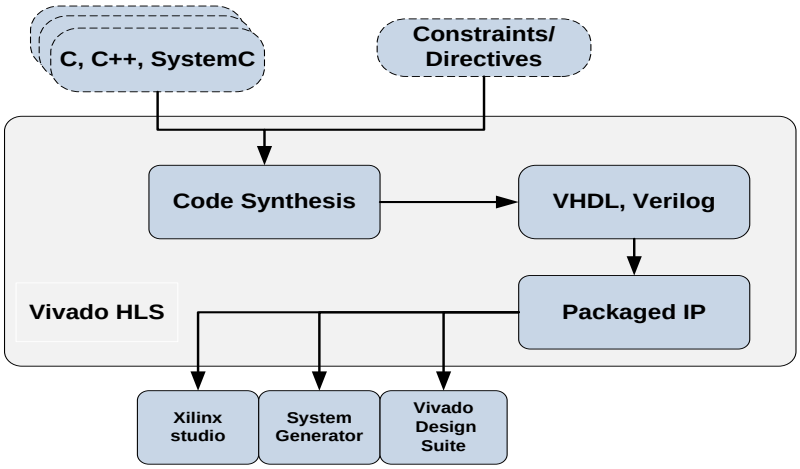


Figure 2.5 Vivado HLS design flow [6]

2.1.1.2 FPGA Architecture

Although FPGA vendors are numerous, the work conducted for this thesis is based on Xilinx FPGAs. This section presents Xilinx FPGAs and Xilinx System on Chips and their architectures. With the evolution of the FPGA industry, the Xilinx company has introduced a number of FPGA families with different features. Xilinx have divided their FPGA families into two main series: the high-performance series and the high-volume series. Before 2010, the Virtex series was the only high-performance Xilinx family and the Spartan series was the high-volume family. After 2010, Xilinx have introduced new families and architectures with 28 nm and 20 nm, which feature reduced power consumption and increased performance compared to the earlier versions. The new families are the high-end Virtex family, mid-range Kintex family, and low-end Artix family in addition to the existence of the old Spartan generation. In addition to the aforementioned families, Xilinx have introduced a new line of production by shifting toward the FPGA/CPU hybrid SoC with the new Xilinx Zynq-7000 all-programmable SoC. This architecture provides increased performance, flexibility, and scalability with the combination of software programmability of the processor and hardware programmability of the FPGA chip. There are different models of Zynq-7000 SoC based on the type of FPGA chip that is attached to the system. This new architecture opens the door for more applications to be fit on such products because the processor is a hardwired part of the chip, unlike

traditional soft processors that are used in FPGAs for embedded solutions. This section presents details of the Zynq SoC architecture.

The architecture of the Zynq-7000 all-programmable SoC consists of two parts: the Processing System (PS) and Xilinx Programmable Logic (PL) [11]. The PS side of the chip consists of a number of elements: Dual-core ARM Cortex A9-based Application Processing Unit (APU), caches and on-chip memory for low latency paths between the CPUs and PL side, external memory interfaces, Input/Output Peripheral (IOP), and the interconnect within the PS elements and between the PS and PL sides. The PL side of the chip is the FPGA itself and shares the same seven-series programmable logics as the Artix or Kintex-based devices. These reconfigurable resources are Configurable Logic Blocks (CLBs), Block RAMs (BRAMs), DSP Blocks, Programmable I/O Blocks, clock management resources, and reconfigurable routing resources. The reconfigurable resources are arranged in a specific order and layout in the reconfigurable space. Each column of the FPGA consists of one type of reconfigurable resources. The logic block presented in the figure could be a CLB, BRAM, or DSP block. Figure 2.6 shows the Zynq-7000 all-programmable SoC and its functional blocks—PS and PL sides [11].

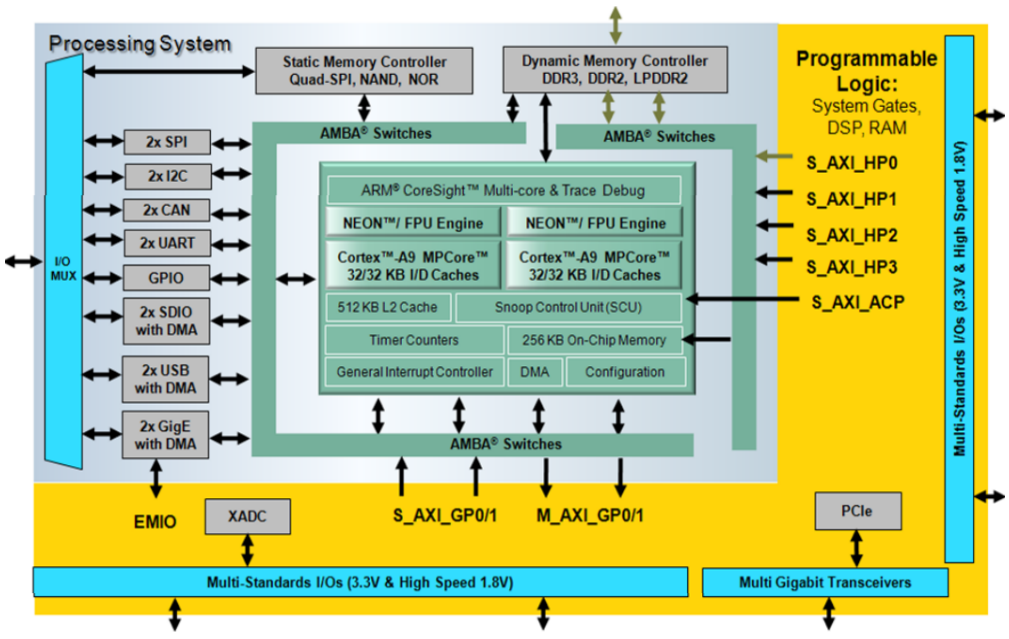


Figure 2.6 Zynq-7000 all-programmable SoC [23]

The existence of processors and surrounding elements on the PS side along with the PL side enables implementation of custom software and custom designs with flexible, scalable, high-performance, low-power, easy to use, and cost-sensitive applications [23]. This architecture can fit a wide range of applications including automotive, broadcast, smart cameras, motor control, and others. Moreover, it enables rapid development and provides more levels of performance and power consumption based on the application requirements.

Programmable Logic Block

The programmable blocks on the FPGAs are the parts that can be configured to build the functionality of the design. The FPGA is divided into vertical regional blocks called clock regions. The height of the clock region is called the column. The following paragraphs describe the main types of the configurable resources in the FPGAs and their architectures based on the Zynq-7000 AP SoC.

Configurable Logic Blocks (CLBs)

CLBs are the main reconfigurable resources of the FPGA. These CLBs are distributed in columns-based in the FPGA fabric. Each CLB column consists of 50 CLBs in the seven-series FPGAs, while other series have different numbers in each CLB column (See Table 2.2). The CLB in seven-series FPGAs consists of two slices with no direct connection between the two slices; the slices are connected to the top and bottom slice in the same column. Moreover, the slices have connections with the switch matrix to access the FPGA general routing. Each slice is organised as a column (see Figure 2.7). Each slice is comprised of four six-input Look Up Tables (LUTs), eight storage elements (flip-flops), a wide function multiplexer, and a carry chain. Other FPGA series have different numbers of components per CLB (See Table 2.2). There are two types of slices: SLICEL and SLICEM. The SLICEM has a few more functions compared to SLICEL. The SLICEM LUT can be configured as a distributed Random Access Memory (RAM) or shift register in addition to look-up table. The distributed RAMs available in SLICEM can be connected and combined in a manner that enables storage of a large amount of data—up to 1MB in some

modern FPGAs. The shift register is configured by delaying the serial data in LUTs up to 32 clock cycles for each LUT and 128 clocks when combining the four LUTs.

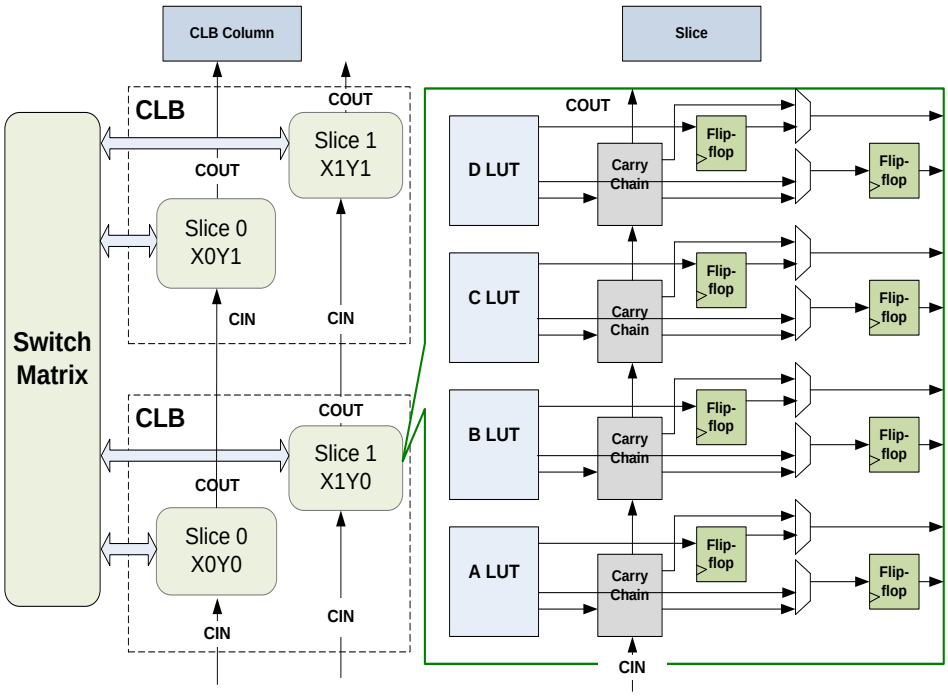


Figure 2.7 Seven-series CLB and slice architecture [24]

Table 2.2 Logic Resources in One CLB for Different FPGA Series and CLBs in a Column

Series	Slices	LUT	Flip-flops	Arithmetic and carry chains	Distributed RAM	Shift Register	CLBs/Col
Virtex-4	4	8	8	2	64 bits	64 bits	16
Virtex-5	2	8	8	2	256 bits	128 bits	20
Virtex-6	2	8	16	2	256 bits	128 bits	40
7- series	2	8	16	2	256 bits	128 bits	50
Spartan-6	2	8	16	1	256 bits	128 bits	16

Block Random Access Memory (BRAM)

Unlike the distributed RAMs that exist in CLBs, BRAM is a special memory component to store large amounts of data, and it offers high speed and customisable memory. The size of each BRAM is 36 Kb, and it contains two independent 18 Kb BRAMs. The BRAM is column-based like the other reconfigurable resources in the FPGA. The BRAMs can be cascaded to implement big memory with low latency. Moreover, BRAMs can be configured as single-port block RAM or dual-port block

RAM. Dual-port memory allows for simultaneous read and write operations. In addition to the previous features of BRAM, it can be configured as First In First Out (FIFO) memory to increase the range of possible uses. The BRAM has an optional error correction circuitry.

Digital Signal Processing (DSP) Slices

FPGAs also have dedicated hard fabric DSP slices that are customised for digital signal processing applications as they contain the functions required for DSP applications. The existence of DSP slices in FPGAs increases the efficiency as the DSP slices are full-custom, low-power, high-speed, small-size, and flexible components [25] . The DSP slices are comprised of the following subcomponents: 25x18 two's complement multiplier for dynamic bypass, 48-bit accumulator for synchronous counters, pre-adder function, single-instruction-multiple data arithmetic unit, logic units for generating different functions for the two-operands, and a pattern detector (see Figure 2.8). The DSP elements are fully customised at run-time through a set of control vectors on the element itself.

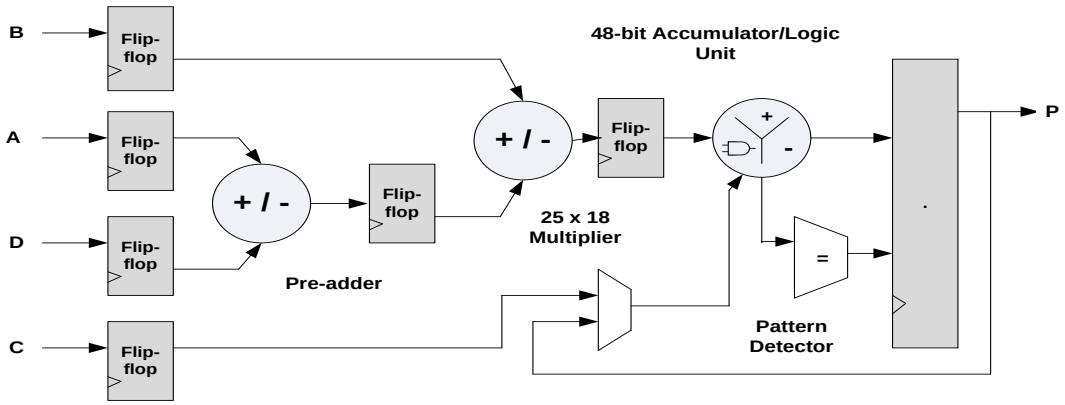


Figure 2.8 Basic DSP slice [25]

Routing and Clock Resources:

The internal routing of the FPGA is called interconnects. These interconnects form a programmable network that connects the elements of the FPGA to each other. The network of interconnect is placed in an optimum way to provide high speed and low latency with fast compile time. The interconnects are connected to the FPGA

elements using a switch matrix for efficient connectivity. The designers cannot control most of interconnects. The features of interconnects are hidden from the users. The clock sources in Zynq all-programmable SoC are categorised into two types: PS clock and PL clock resources. The PS clock is generated using three Phase-Locked Loops (PLLs), as there are three primary domains: APU, Double Date Rate (DDR), and I/O peripheral. Different frequencies can be configured under software control. The PS clock can be directed to feed the PL side of the Zynq AP SoC (see Figure 2.9).

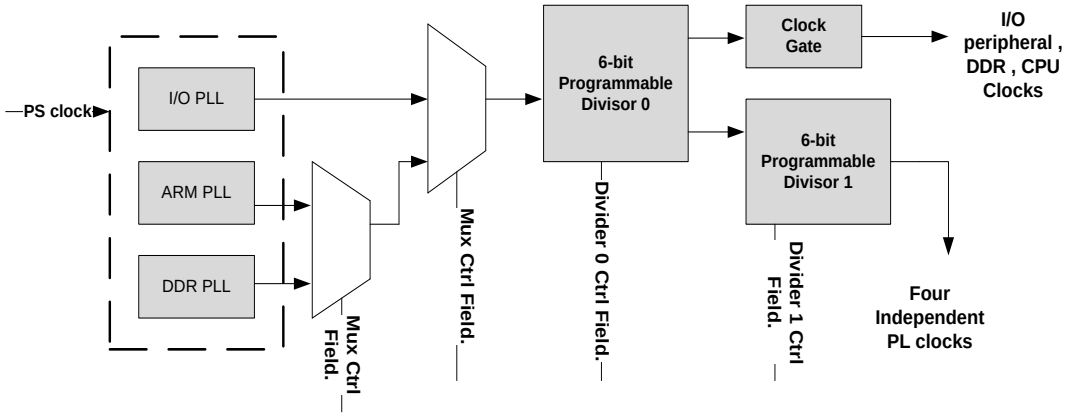


Figure 2.9 Basic PS clock diagram [11]

The PL has its own clock management in addition to the four clocks received from the PS side. The FPGAs need the clock sources to be connected to the internal resources. Each bank has four clock-capable inputs. These could be single-ended or differential clock. Moreover, the seven-series FPGAs have up to 32 global clock lines with high fan-out to reach every single component in the die of the FPGA. The FPGAs are divided into clock regions. Every clock region has twelve global clock lines. Additionally, every clock region has four dedicated regional clocks to feed the element of that region and the two adjacent regions. These clock resources can be divided by integers from one to eight for the different frequencies. Another clock resource is the Clock Management Tile (CMT), which consists of one Mixed Mode Clock Manager (MMCM) and one PLL. These elements are used as synthesiser for the wide range of frequencies and as jitter filter.

FPGA resource categories:

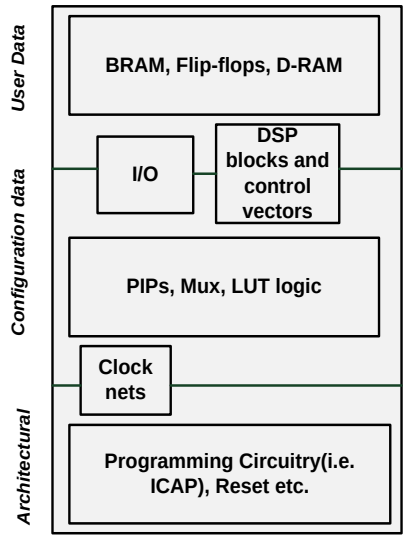


Figure 2.10 FPGA resources groups [26]

In [26], the work showed that the FPGA resource elements can be categorised into three main groups based on the type of configuration memory content: configuration data, user data, and essential architectural data. The first group, configuration data are mainly the group that defines the logic functions and the routing paths of the implemented design such as LUT content, PIP connections and the signals that build up the multiplexers. These determine that overall functionality of the design and any changes in these configuration memories shall change or corrupt the design. The second group, user data are the content of the storage elements dynamic memory. The user can change this content during normal operations depending on the operations executed by the design. It is normally the content of flip-flops, BRAM, distributed RAM synthesised as LUT, and the Shift Register Lut (SRL). This group determines the processing data in memory and dynamic operations that changed with the time in the design. The third group, essential architectural data are the essential data for the operation of the FPGA itself such as programming logic, clocking management elements, and reset circuitry. The design cannot work if the first and the third group contents are changed while it can work fine if the second group content is changed. The majority of bits are belonging to the first group of data with approximately 80% of the bits. Figure 2.10 shows the groups and the corresponding

FPGA elements. Some of these elements are resided between two groups because its configuration memories are distributed between the two groups.

2.1.2 Dynamic Partial Reconfiguration

The flexibility factor is highly related to the FPGA technology because of the capability of on-site programming without the need to go through the re-fabrication process. This can be achieved by loading the static configuration, which is the entire FPGA configuration information in a file called bitstream. To configure this bitstream, all operations on the FPGA should be halted while the new bitstream is being configured. The manufacturers of the FPGAs are pushing the flexibility FPGA technology one step forward by introducing a technology that enables the user to configure part of the FPGA chip that has a specific function while the other parts with other functionalities are functioning. This technology is called DPR. DPR feature allows for sharing the hardware resources of the FPGA over time, which allows for configuring unlimited number of tasks with no limited times in a specific portion of the FPGA by swapping the tasks in and out. Applying DPR in this way allows for reducing the required hardware resources for specific function, reducing the total power dissipation, increasing the system performance, increasing the system reliability, and increasing the system flexibility. In contrast, the static design has a limited flexibility and cannot provide such features to the implemented functionality. Technically, the implementation of any design using DPR should have a full bitstream file as the normal FPGA implementation. At power-up time, the full bit file has to be configured in to load the full functionality on the FPGA as the FPGA technology cannot keep the configuration data after power-off because of the volatile SRAM technology. After the full bitstream is loaded, the partial bitstreams can be loaded internally or externally at any time to alter the functionality of part of the design through one of the configuration ports (see Figure 2.11). The configuration port passes the data to specific configuration memory address.

When this technology was introduced, only the high-end FPGA devices supported the feature. Presently, most of the FPGA devices from different manufacturers support DPR feature to increase the range of FPGA applications. Any design that

uses DPR feature must be divided into two types of partitions: static partition or region and Reconfigurable Partition (RP) (see Figure 2.11). The static partition is the part that does not change during run-time. The clocking resources should reside in the static region with exception of the ultra-scale devices [27]. The RP is the part that holds the Reconfigurable Modules (RMs), which are swapped in and out at run-time. RMs contain the actual hardware resources that represent a specific functionality. The example in Figure 2.12 shows four different RMs with different bit files, one for each module. Any RM can be configured in the specified RP by swapping in the configuration file after stopping the operations of that specific task and isolating the entire task from the surrounding components. The configuration file should be stored in a place outside the FPGA logic. A configuration controller should be involved in the process to fetch a specific file from the external memory and allocate it to a specific region. At the time of implementing the DPR design, the user has no control over the routes of the static partition. These routes could pass through the RP; hence, any RM corresponding to that partition should contain these routes with the same shape. The DPR tools should maintain the interface between the static partition and the RP because they should be identical to each other. On the other hand, all RM routes should be capsulated within the RP. No local routes are permitted to reside outside the RP. In the design flow, and based on experiments, the assigned RP resources should be approximately 20% more than the RM resources for routing purposes and placing PR related components.

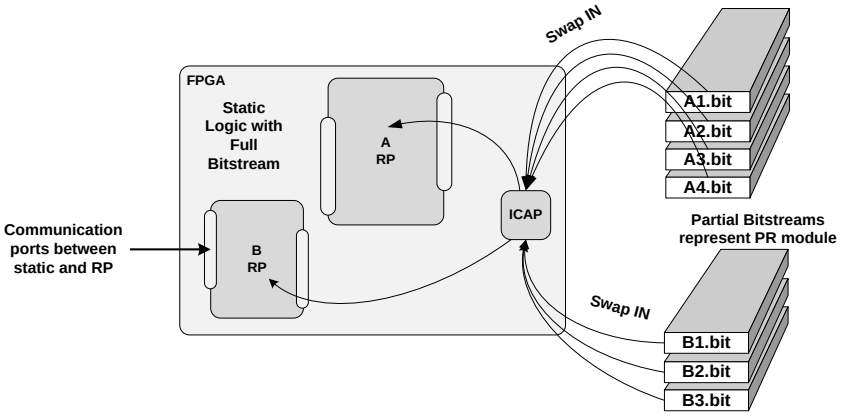


Figure 2.11 Partial reconfiguration example in FPGAs

2.1.2.1 ISE and Vivado DPR Flow

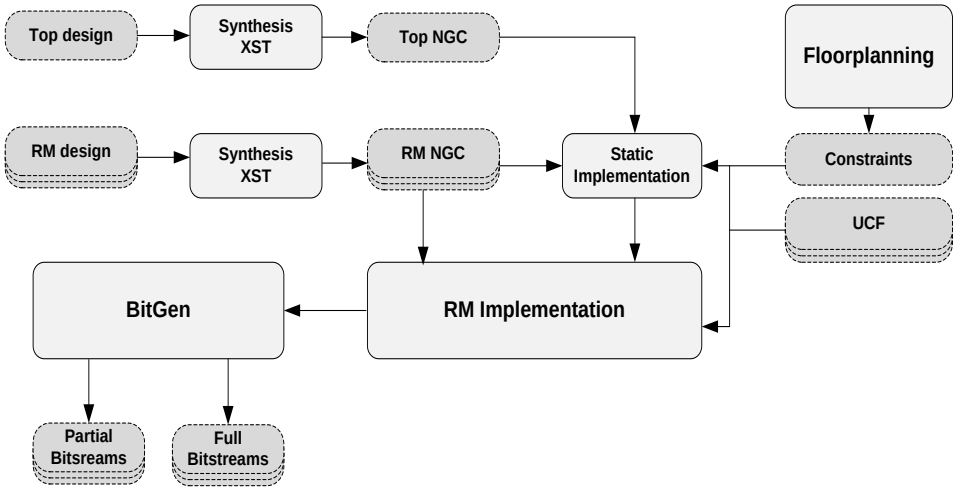


Figure 2.12 Simplified Xilinx ISE DPR design flow

Xilinx have two set of tools that support partial reconfiguration. A few years ago, PlanAhead, part of the ISE design suite, was the only Xilinx tool that supports PR. ISE does not support the new Xilinx seven-series devices. Today, Vivado design suite supports the feature in this series. The two tools have different DPR design flow. Figure 2.12 shows the DPR design flow using the ISE tool. Alternatively, Vivado design flow has similar standards with few differences. The software automatically manages the low-level details. Vivado has no specialised Graphical User Interface (GUI) for DPR. Vivado employs memory-based flow with no project file to save the work. This flow, known as non-project based flow, it can go through the entire flow by running each design step individually with save feature.

The DPR Vivado design flow (see Figure 2.13) should proceed in the following steps. First, the user should build-up the entities of the design with clear separation between the static and the RMs. The modules should be synthesised separately to create different Checkpoint files (.dcp). On the synthesised top-level module, physical constraints should be created to define the I/O constraints and the reconfigurable regions with 20% more resources compared to the synthesised RM. The RP should set the HD.RECONFIGURABLE property to activate the PR feature on that partition. After that, one RM should be assigned for each RP to implement

the full design by issuing the following commands: opt_design, place_design and route_design. The design checkpoint for the full design routes should be saved in a file to be used in generating the bit files latterly. At this point, only one RM is implemented. The routes of the static partition should be identical among all the implementations. To do that, the assigned RM should be removed, and the design checkpoint of the static routes should be saved to be used in each implementation. The lock command should be issued to lock the routes on the chip. The following steps should be repeated for the next RMs:

1. Assigning the new RM
2. implementing the full design
3. Saving the design checkpoint
4. Removing the RM

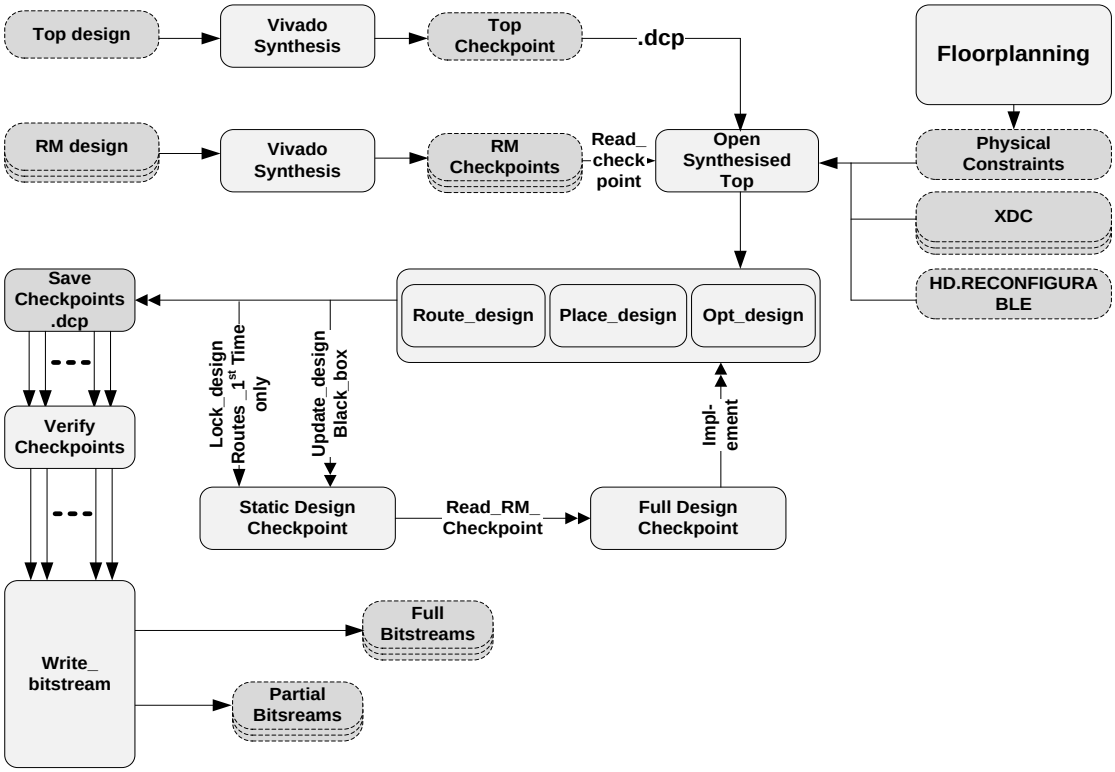


Figure 2.13 Simplified Vivado DPR design flow

After that, all design checkpoints should be verified using a verification utility to check the routes among all implementations. The bit file can be generated after the verification process using write_bitstream command. Two sets of bitstreams will be

created: full bitstreams and partial bitstreams. To reduce the power consumption, Xilinx provided the possibility to assign an empty 'black_box' to the RP.

2.1.2.2 Configuration Frames and Ports

The configuration memory in an FPGA is a set of volatile memory cells that should be configured to manage the FPGA resources. These cells are arranged into segments known as memory frames. The frame is the smallest addressable segment of the device configuration memory [27]. In the DPR, the smallest reconfigurable region is decided based on the frame size. The size of frame varies depending on the FPGA family. In seven-series, for CLB resources, the frame is 50 CLBs high by one wide. The DSP frame is 10 DSPs high by one. The BRAM frame is 10 x 1 BRAMs.

Partial Reconfiguration can be performed externally using one of the external ports or internally using the ICAP or the Processor Configuration Access Port (PCAP) in Zynq devices. Different ports are used with different specifications. Table 2.3 shows the specifications of different configuration ports. The ICAP needs special reconfiguration control logic to control the data in and out from the port. Two ICAPs are available in the FPGA, but only one can be used at a time, with the possibility of switching between them using a special switch command. In contrast, the PCAP is a ready connected port in the Zynq devices. The PCAP can be controlled directly from the software. The ICAP is faster than the PCAP because of the limitation in throughput of the PS AXI interconnects [11]. The configuration speed depends on the size of the bit file and the bandwidth of the selected port. Table 2.3 shows the bandwidth of different ports.

Table 2.3 FPGAs Configuration Ports

Configuration port	Port Type	Max Frequency (MHz)	Data width	Bandwidth
JTAG	External	66	1 bit	8.25 MB/s
ICAP	Internal	100	32 bit	400 MB/s
SelectMAP	External	100	32 bit	400 MB/s
Serial Mode	External	100	1 bit	12.5MB/s
PCAP	Internal	100	32 bit	145 MB/s

2.1.3 Dynamic Partial Reconfiguration Deployments

The high performance of FPGAs comes from achieving more computation parallelism through optimal use of the resources in the FPGAs. Traditionally, single-core or multi-core CPUs have been the main sources of computation power. Servers and supercomputers used for such purposes cannot reach the performance of hardware platforms. With the evolution of FPGA technologies, FPGAs have become an attractive environment for engineers to develop a wide range of applications. In recent years, manufacturers are targeting ASIC-like devices that compete with ASIC technology in terms of performance and power consumption based on new clocking and interconnects technologies. Additionally, FPGA technology already presents an advantage over ASIC technology because of their flexibility. DPR increases the flexibility of the FPGAs one step forward by changing the functionality of subsystems without interrupting the whole system. This can be used for hardware acceleration purposes in which a number of accelerators can be changed on demand, based on the system nature. Many research efforts discuss involving DPR in applications because of its impact on performance, power consumption, and area.

2.1.3.1 DPR in High Performance Systems

The term High Performance Computing (HPC) refers to the case in which the application demands have outpaced the ability of conventional processors [28]. FPGAs, as mentioned previously, are an attractive solution for such applications. The FPGAs can be deployed in high-performance applications in different ways. An FPGA can be used as a bridge or switch between different interfaces or subsystems with different connectivity standards, protocols, or logic. The low power, high flexibility, and high data rate of modern FPGAs make them ideal for such applications. In [29], the white paper from Lattice Semiconductor shows a few examples of FPGA bridging applications. In [30], a large Network-on-Chip (NoC) based system is partitioned into smaller subsystems with different protocol levels. The FPGA here acts as a bridge between subsystems to preserve the quality of the system. FPGAs also can be used as fixed hardware accelerators when high data throughput is required from a specific function. The function elements can be

executed in parallel to provide high data throughput. In [31], the beam-forming algorithm, which is the data-intensive component of the high throughput radar, is accelerated in FPGA. In this context, many algorithms are accelerated in FPGAs to obtain the desired data throughput, especially in real-time applications. Examples of different accelerators in various fields are provided in [32], [33], [34], and [35]. Moreover, FPGAs can be used in software acceleration by moving part of the software application from the CPU to the FPGA to compensate for the shortage in processing in some cases or to increase the overall performance. In this context, the manufacturers of FPGAs established a new line of production by shifting toward CPU/FPGA SoC to make the software acceleration possible in one environment with low-cost development. The Zynq-7000 AP SoC from Xilinx and the Altera SoC are the current hybrid SoCs available in the market. Examples of FPGA co-processor applications in which part of the application is processed on the FPGA in parallel and the remaining part is processed in the CPU with a reliable connection between the two parts are provided in [36] and [37].

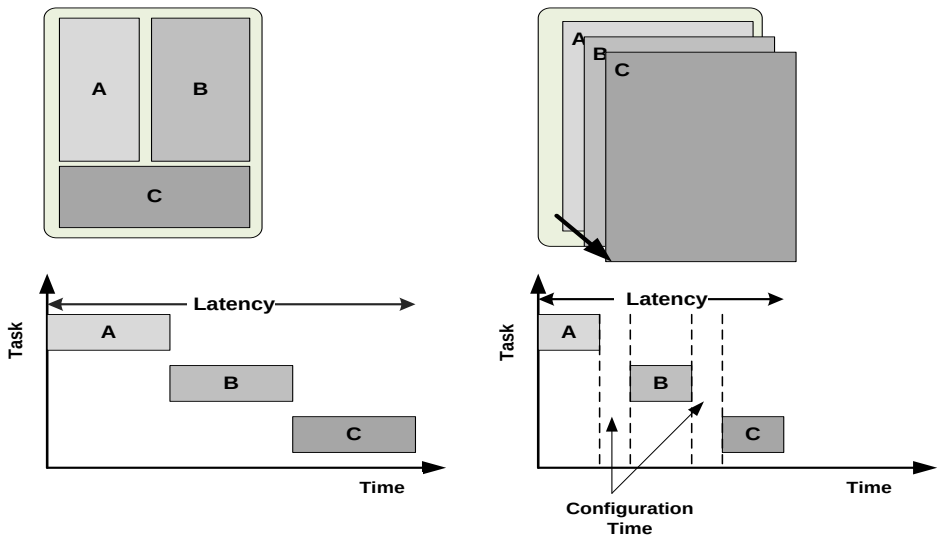


Figure 2.14 Enhanced system performance with DPR [38]

Deploying DPR can enhance the performance of the system compared to the static implementation in some cases. The author in [38] shows one of the cases in which the system has a number of tasks that are executed sequentially. The idea is to share the resources of the FPGA between the tasks. Time-multiplexing the resources of the

FPGA means that more resources are available for each task-module, which can increase the execution parallelism and which leads to a reduction of the execution time for each task as well as the total execution time for the system. Figure 2.14 shows the difference between the static and the DPR implementations and their impact on the execution time. DPR implementation latency is less compared to the static one due to the total computation power available in the two cases.

The SRAM-based FPGAs store the configuration data in the static memory and cannot keep the data without a power source. Because of that, the FPGA has to be programmed upon start. With the increase in FPGAs density, the total configuration data is enormously increased. Therefore, the configuration time upon start is an issue for some applications that demand fast availability. In the case of high-density FPGAs, the configuration time could be long enough to create a problem for the running application because of the initial configuration using slow flash devices. DPR can solve this issue by bootstrapping, where part of the system—the smallest part of the system that requires fast availability—is loaded first, and then the remaining part is configured in a further step. This solution helps the systems that demand fast availability to work efficiently with high performance from the beginning. Bootstrapping is discussed in details in [39] and [40].

The use of DPR to enhance the system flexibility or performance requires continuous use of configuration ports to configure partial bitstreams. The configuration time overhead is a bottleneck for high-performance applications that swap tasks in and out continuously. A few techniques are discussed in literature to speed-up the configuration process to meet the high-performance application demands:

Reducing configuration time overhead: The bandwidth of the configuration port determines the speed and configuration time of the partial bitstream. Technically, the maximum possible bandwidth is 400 MB/s through the ICAP and SelectMAP ports. To utilise the full available bandwidth of the port, the bitsream should be stored in high-speed memory after power-up to eliminate any delay in the path between the ports and the location of the bitstreams. Usually, the bitstream should be stored in SRAM or DDR memory. In addition, a high-speed controller should be implemented

in the logic or software to move the data efficiently from the memory to the configuration port. Various types of such controllers exist. The Xilinx HWICAP IP core ([41] and [42]) controls the flow of data between the memory and the configuration port and vice-versa using either soft-processor or hard-processor. This method is an inefficient way for high-performance applications because the processor will be busy constantly controlling the flow of data. Moreover, it provides low throughput compared to the capability of the configuration port. In [43], the author proposed an ICAP controller that uses a big memory constructed through the on-chip BRAMs to hold the entire bitstream. This method is insufficient, as the bitstream size will be limited to the available memory. In [44], the author proposed an ICAP controller that uses Direct Memory Access (DMA) for moving data, and the processor is only responsible for initiating the transfer operation by sending commands to the DMA controller. This method reached a bandwidth of 392 MB/s. Other controllers in the literature utilized the full speed of the ICAP primitive such as the controller proposed in [45] and [46] and the ZyCAP proposed in [47] which is implemented in Zynq-7000 AP SoC.

Compressing the bitstream is an effective way to reduce the total fetching time from memory and therefore the total configuration time. The Multiple Frame Write (MFW) feature from Xilinx is considered a way of compression by writing the redundant configuration frames once instead of configuring them individually. This method will reduce the configuration time mostly in the case where the design is not utilising the resources of the FPGA. The compression scheme proposed in [44] achieved up to a 75% reduction in the bitstream size.

Over-clocking the ICAP primitive: Although this way is not recommended, it can achieve higher bandwidth compared to the maximum bandwidth mentioned in the manufacturer's data sheets. Exceeding 100 MHz, the maximum frequency could lead to unpredicted ICAP behaviour under different conditions. A few works in the literature have discussed this and successfully managed to over-clock the frequency of the ICAP primitive with reliable configuration attempts. In [48], the work reported 144-MHz maximum frequency in the Virtex-4 FPGA. In [49], the author reported a

successful over-clocking attempt with a frequency up to 362.5 MHz in the Virtex-5 FPGA and several MHz lower in the Virtex-6 FPGA.

Pre-fetching RMs: In the case of multi RMs and Multi RPs where the modules have to be processed individually, the RM can be configured before its time while the other module is executing. This method will mask the configuration time of the follower modules. This method is discussed in [50], which managed to enhance the throughput by 29.9% compared to the normal method. The work carried in this paper is suitable for a known sequence of RMs. In the case of multi RMs in which the next module to be configured is unknown until the last moment, this method is still applicable but with additional scheduling algorithms that predict the next module to be configured. The combination of pre-fetching and the scheduling algorithm increases the performance of the Reconfigurable OS in FPGA, which is becoming a trend in recent years. Figure 2.15 shows the pre-fetching technique with several RMs.

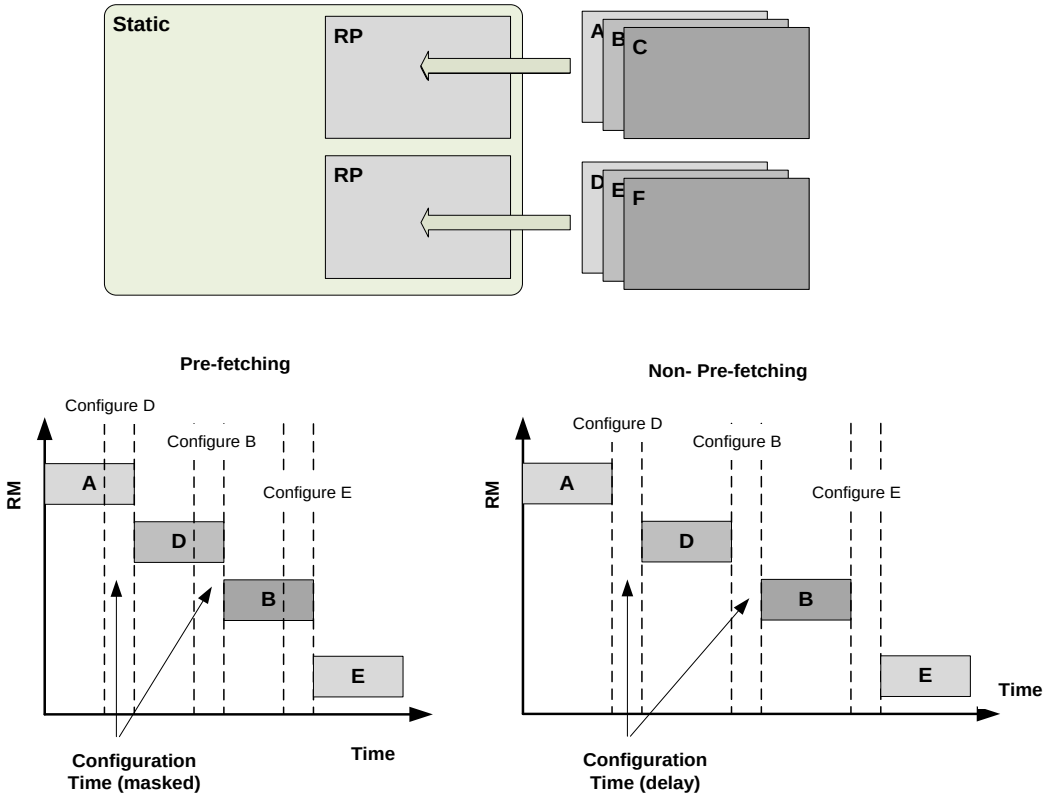


Figure 2.15 RM pre-fetching (virtual reconfiguration)

2.1.3.2 DPR for Power and Area Saving

The power consumed by FPGA is divided into three components [51]: static power, dynamic power, and I/O power. Static power is the power consumed when no signals are toggling. The power is consumed in leakage form. The dynamic power is the power consumed when the system is operating, the signals are toggling, and the capacitors are charged and discharged. The dynamic power is directly affected by supply voltage, clock frequency, effective capacitance, and switching activities [52]. The last component of power is the I/O power, which is consumed by the general purpose I/O and serial transceivers. Approximately 60% of the total consumed power is dynamic and I/O power [51]. DPR can reduce the power consumption by reducing the density of the system in the FPGA, enabling the use of smaller FPGAs that consume less static power and therefore less dynamic power by loading only the part of the system, which has the functions that operate simultaneously, and configuring the other parts in later stages (See Figure 2.16). Another practice of DPR to reduce the power is proposed in [53]. The tasks on the FPGA can be fed with different frequencies and therefore different levels of power consumption. The high frequency partitions consume more power than low ones. In the case of tasks that are not time-critical, the frequency can be reduced to minimum or a certain level to conserve the power. Moreover, DPR can reduce the power in a different way by removing the configuration data of inactive regions and converting them into black boxes, which have no logic and therefore no dynamic or I/O power consumption.

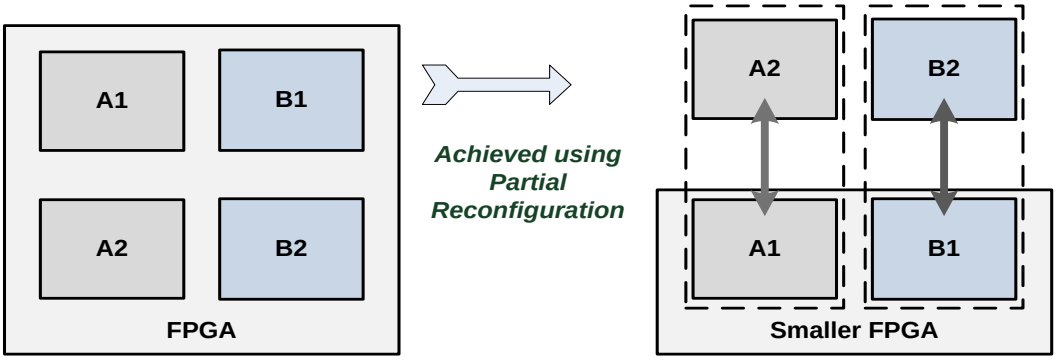


Figure 2.16 Enabling power consumption by using smaller FPGAs [51]

The DPR feature allows the hardware resources of the FPGA to be time multiplexed to perform different operations over time. This enables the possibility of implementing large systems in a small area footprint, which has less resource than those needed by the system. This practice is widely used in the literature to allow the engineers to build a wide range of applications in a smaller area without any limitations (See Figure 2.17). In [54], DPR software defined radio implementation is presented. The work has more than 49% average reduction in area costs compared to the static design. Many works in the literature discuss the same concept, particularly the use of multiple algorithms for the same function. The static design for such applications must handle all algorithms in the design footprint, which results in high resource cost. The DPR implementation can save a significant amount of resources by handling only one algorithm -module while the others are kept in the memory and configured upon need (See Figure 2.17).

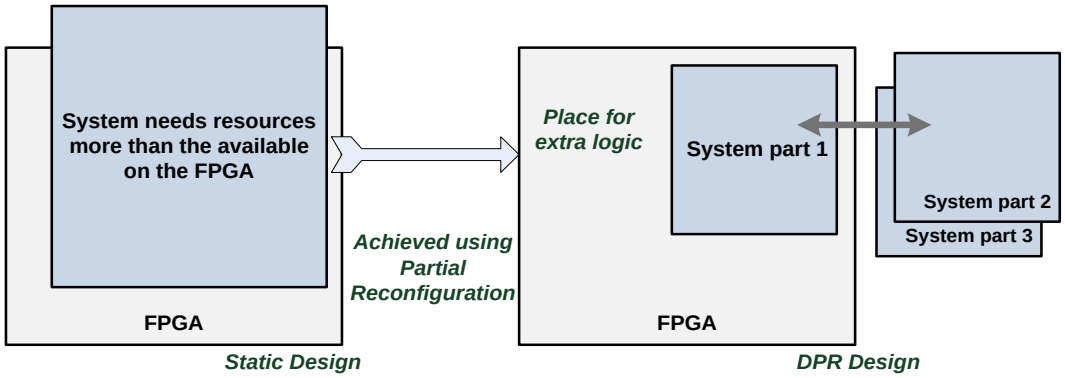


Figure 2.17 Area optimisation using DPR

2.1.4 Reliability in FPGAs

FPGAs are widely used in the industry applications. However, the SRAM-based configuration memories used in FPGAs are highly sensitive to radiation. Radiation can cause corruption to the data stored in the configuration memory, which leads to functionality faults in any part of the implemented system. With the increase of configuration cell density in FPGAs, the probability of faults occurring is increased. This section summarises the types of faults in modern FPGAs and presents the

techniques available in the literature or embedded in the FPGAs that make the SRAM FPGAs more reliable.

First, it is important to express the system reliability quantitatively to give an appreciation of how the system can perform in a specific environment with specific properties. This can be achieved by using a probability density function to describe the probability of an event -failure in this case- as a function of time. There are many terms to describe the reliability based on methods and procedures for product life cycle prediction such as Failure in Time (FIT), Mean Time to Failure (MTTF), Mean Time to Repair (MTTR), Mean Time between Failtures (MTBF), and Failure rate. In the case of MTTF, it is the mean time expected for the first failure to occur in the logic. This again can be expressed using a probability function as shown in Equation 2.1 [26].

$$MTTF = E(X) = \int_{-\infty}^{\infty} x\, f(x)\ dx \qquad (2.1)$$

MTTR is the time required for repair, reset, or replace the faulty unit, while MTBF is the rate of faults to be occurred and repaired. It is defined as shown in Equation 2.2. Also, it can be expressed as the ratio between reliability function and the probability density function as shown in equation 2.3. Failure rate λ(x) is the rate of failures per unit of time and can be expressed as an MTBF-based function as shown in Equation 2.4

$$MTBF = MTTF + MTTR \qquad (2.2)$$

$$MTBF = \frac{R(x)}{f(x)} \qquad (2.3)$$

$$\lambda(x) = \frac{1}{MTBF} = \frac{f(x)}{R(x)} \qquad (2.4)$$

The availability of the system is represented by the total system uptime compared to the total system including the downtime. The availability is shown in Equation 2.5. FIT is the number of faults per billon hours operation per device and is calculated as shown in 2.6

$$\text{Availability} = \frac{MTTF}{MTBF} \qquad (2.5)$$

$$\text{FIT} = \lambda(x) * 10^9 = \frac{10^9}{MTBF} \qquad (2.6)$$

2.1.4.1 Types of Faults in SRAM FPGAs

The fault can be defined as a physical defect, imperfection, or flow occurring in hardware or software components. The faults in SRAM-FPGAs are classified by their duration into three types: permanent faults, transient faults, and intermittent faults [55]. The permanent faults are caused by irreversible damage in the device components such as conductor's ions movement, which leads to a short circuit fault. The permanent fault can only be recovered by replacing or repairing the damaged part. A few phenomena can cause such faults, including electromigration [56] and Hot Carrier Effects (HCE) [57]. Electromigration can cause faults by producing heavy current densities on the interconnect over the period. Intermittent faults can be caused by unstable hardware or varying hardware states [55]. This type of fault can occur at the same location many times, and it can be a sign for a future permanent fault. The last type of fault is the transient fault, which is known as a soft error. This is a temporary error triggered by environmental conditions such as electromagnetic interference, alpha particles, and cosmic ray neutrons ([58] and [59]).

Transient errors are changes in the status of the device storage elements. They can be fixed by changing back the status to its original value using special detection and correction techniques. Soft errors are classified into many types: Single-Event Transient (SET), Single-Event Upset (SEU), Multiple Bit Upsets (MBUs), and Single-Event Latchup (SEL). SET mainly appears as glitches, which are unexpected outputs caused by different propagation delays for the combinational logic resources that are involved in the path in the system [60]. These glitches can cause faults if they are captured by flip-flops or memories. SEU and MBUs are revealed as single bit-flips in the configuration memory cells. SEU is a single element state change, while MBUs is multiple upsets in the logic from a single radiation strike [59]. Mainly, the upsets are affecting logic, I/O, or routing configuration memory bits. The rate of how susceptible the chip is to soft errors is expressed by the two terms: MTTF and FIT. FIT is the number of failures in a billion hours of operation. Over multiple generations of Xilinx FPGA series, the FIT rate is changed based on the transistor technology and the density of the chip (see Figure 2.18). The rate is expressed by X

FIT/MB (Where X number of faults per billion years per 1 Megabits of configuration data). However, not all the upsets in the FPGA could lead to a functional error because no design ever uses the capability of every element in the entire chip [61].

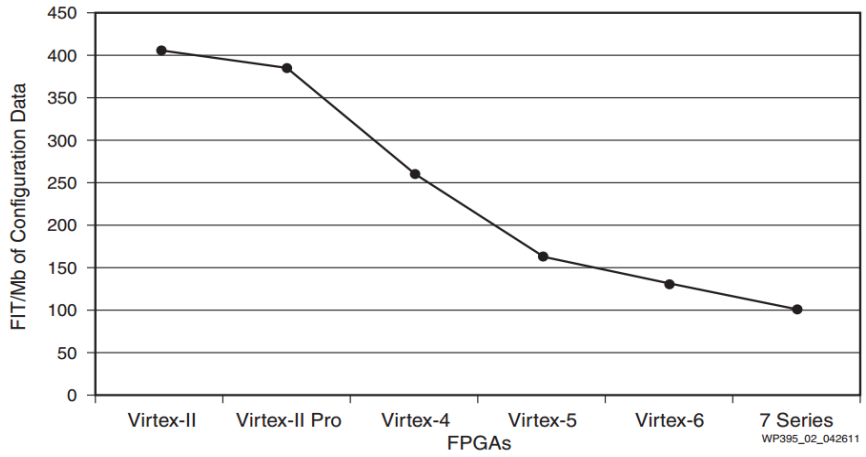


Figure 2.18 Xilinx FPGA failure rate by product [61]

2.1.4.2 Towards Reliable FPGAs

FPGA manufacturers have added features to their FPGAs to mitigate soft errors and minimize the failure rate. In addition, a few techniques that deal with such issues are available and discussed in literature in detail.

Reliability features in FPGAs: A special production line is dedicated to fabricating hardened FPGAs that are specialised for space and high-radiation environments. Alternatively, commercial FPGAs are using built-in detection and correction methods such as Error Correction Code (ECC) and Cyclic Redundancy Check (CRC) to minimise the FIT rate. Each configuration frame has ECC parity bits to determine the location of the flipped bit in the frame. The length of parity bits varies among different FPGA series based on the size of the configuration frame. The ECC code is calculated at the time of generating the data frame by the bitstream generator tool. Virtex-4 contains a 12-bit Hamming code, as the frame consists of 1312 bits. These bits are known as Syndrome. Virtex-6 and 7 series have a 13-bit Hamming code, as the frames consist of 2592 and 3232 bits, respectively [62]. The Hamming code can be used to detect up to two errors and correct one error in each frame. Xilinx has a

dedicated hard-wired ECC logic in their FPGAs to detect any bit-flips in the configuration memory frames during the read-back operation. Any fault detection in during the read-back, the ECC logic uses the 13-bit syndrome to acknowledge the fault and the place of the fault within the frame. Table 2.4 shows the possible syndrome codes and the meaning of each code for different series. The ECC logic has a dedicated ECC primitive to output the syndrome code and the error signal.

Table 2.4 ECC Syndrome Codes for Virtex-6 and 7 Series

Syndrome (12)	Syndrome (11:0)	Meaning	Detection	Correction
(12) = 0	(11:0) = 0	No bit errors	-	-
(12) = 1	(11:0) != 0	One bit error	Yes	Yes
(12) = 0	(11:0)!= 0	Two bit errors	Yes	No
Unpredictable syndrome		More than two bits	No	No
(12) = 1	(11:0) = 0	Parity bit error	Yes	No

CRC is used in FPGA as a verification parameter at the time of configuration. A predefined CRC code is compared to the calculated code using the CRC logic in the board to detect any fault in the configuration process. Moreover, it is used as detection and correction code for the entire FPGA configuration or for individual frames. The CRC code can correct more than one bit, unlike the ECC code.

State-of-Art Soft error mitigation techniques:

Before discussing the methods, it is important to know that not all the upsets in the FPGA configuration memory can lead to a functional failure because not all the components of the FPGA are fully utilised by the design. Some of the faults in the design can appear in a form of wrong output while the design is working properly. Therefore, not all techniques can help in all cases and all times. Generally, the soft error can affect either the design behaviour or the design state [63]. In the case of the design behaviour, this happens when a soft error occurs in the device configuration memory such LUT while the error in the design state happens if the soft error occurs in memory element such as BRAM, distributed memory, or flip-flops. The techniques of error mitigation depend on either the content of the configuration memory or the output of the processing function. Two main methods have been

discussed widely in the literature for mitigating soft errors: rewriting the frames or scrubbing and modular redundancy. The two methods use the DPR feature to obtain the best outcome. Scrubbing is the process of quickly repairing the upsets before they can accumulate in the memory. The repairing process is carried on the frame through the ICAP by reading the configuration frames and determining if any changes need to be done on the frames and finally writing back the frames to the configuration memory. The scrubbing is categorised into two types: internal scrubbing and external scrubbing. Internal scrubbing utilises the parity bits in each frame to detect and correct the soft errors. This process is carried totally through the internal components of the FPGA. The process is performed in several steps: reading the frames using the internal configuration port, and examining the output of FrameECC primitive logic at each frame readback attempt to detect and correct the errors based on the information shown in Table 2.5. The frame is modified online on the chip and then the frames are written back to the memory using ICAP (See Figure 2.19). However, not all the resources are covered by the scrubbing scheme. Soft Error Mitigation (SEM) IP ([63] and [64]) is a pre-verified IP core that detects and corrects soft errors using ECC bits and CRC code. It is provided with a full capability to manage the mitigation process and an injection feature for testing purposes. The IP classifies the bits into essential and non-essential bits to determine whether the soft error will affect the functionality of the system. Many works in the literature discuss this type of scrubbing, including [46], [65], and [66]. Conversely, external scrubbing does not use the parity bits embedded in the configuration frames but uses a reference golden bitstream kept in non-volatile memory outside the chip. The process is accomplished by a 'read and compare' mechanism between the readback frame and the golden copy. Any time a difference is detected, the equivalent golden frame overwrites the readback one. Moreover, in some cases, a blind scrubbing is done by overwriting the frames continuously without any compression step. The external scrubbing can perform much better than the internal scrubbing as it can correct an unlimited number of soft errors. The external scrubbing feature is embedded in SEM IP core [63].

Modular redundancy is a different mitigation concept. This concept requires more than one copy of the hardware module to be implemented on the FPGA. The

hardware modules should work together and process the task at the same time. Any difference in the output among the different modules indicates error detection. This concept is much stronger than scrubbing because all the logic paths, not only the flip-flops, are susceptible to SEUs [67].

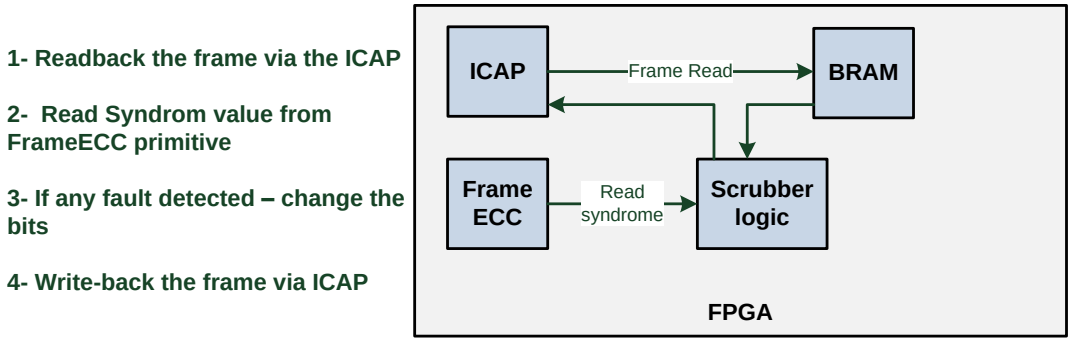


Figure 2.19 Internal scrubbing via ICAP

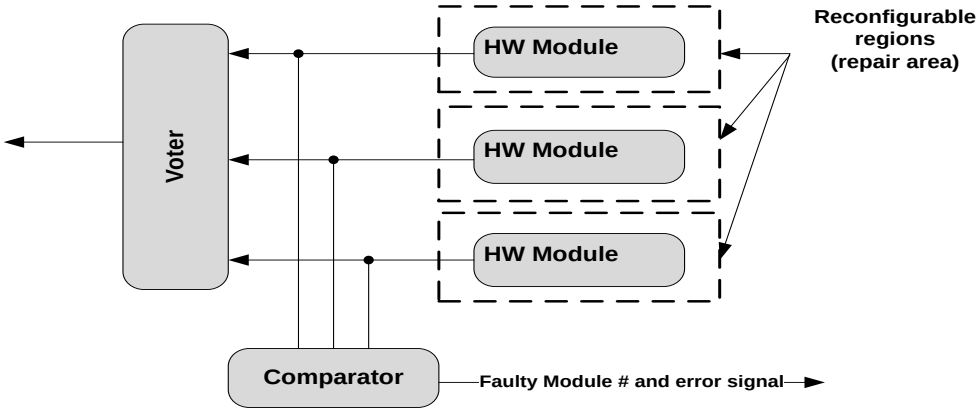


Figure 2.20 TMR scheme for soft error mitigation

More than one type of modular redundancy form is described in the literature. Triple-Module Redundancy (TMR) is the most common form of redundancy (see Figure 2.20). It needs three hardware modules to be placed in the FPGA to generate three outputs. These outputs are passed through voter logic to detect the failure module if exist and trigger the design to pass only the correct output and recover the faulty module [68]. The problem in TMR is the high resources overhead required by triplicating the module in the FPGA. Dual-Module Redundancy (DMR) is another form of redundancy; it is used for error detection with no possible correction criteria

because only two outputs are available at the voter side of the design. Moreover, it is used for reducing the resource overhead compared to the TMR [69]. Time redundancy is a different type that exploits the time domain rather than the physical domain by repeating the operation many times and comparing the output each time. This technique will reduce the resource overhead significantly compared to DMR and TMR. In the case of redundancy, each hardware module should be implemented in a different reconfigurable region for recovery purposes. The faulty module will be recovered by configuring the module partially using the partial bitstreams, which are stored in the memory. Based on the information mentioned at the end of section 2.1.1.2, not all the configuration memory content change will change the system functionality or lead to system corruption because some of these contents are user data information. These configuration memory data have a relation with the design operations over time. Because of that, the content of these configuration memories are usually masked by bitstream generating tool at the generating time, which prohibit the user of changing them for testing purposes such as FFs content. In this context, the scrubber-based mitigation technique cannot discover all type of upsets in the configuration memory because some of them are masked bits.

2.2 Introduction to Image Processing Systems

Digital Image Processing (DIP) is the process of using computer algorithms to perform image processing on multi-dimensional digital images. It is a subcategory of DSP. Due to the digital characteristic of the technology, DIP allows a much wider range of algorithms to the input data, unlike analog image processing, which can build up noise and distort the signal during processing [70]. Moreover, it allows the use of more complex algorithms that are impossible to be built in analog form. Digital images are composed of columns and rows. These rows and columns are intersected to form the pixel, which is the smallest element in the image that holds quantised values that represent the colour information at that point. Generally, image processing algorithms are computationally intensive due to the huge amount of operations required to be done for each pixel or group of pixels in a single image. In [71], the author classified the image processing algorithms into three main levels

based on the required amount of data processing (see Figure 2.21). The low level of the pyramid, the pre-processing, is the most data-intensive part, as this part works at the pixel-level (i.e., the algorithm part of the image-processing pipeline, filtering, thresholding, scaling, or linear scaling algorithms). The intermediate part is focused more in extracting features from the image and transforming the image into something different that can be a representation of a description or decision, such as the classification and segmentations algorithms. This part requires less data processing compared to the lower level. The top level has the least data processing demands, as this part deals with objects such as recognition algorithms. Each image-processing algorithm has its own execution and memory access demands. Therefore, different strategies should be considered at the time of implementation. In this context, not all the platform technologies are suitable for image processing algorithms because each algorithm needs a specific level of computation.

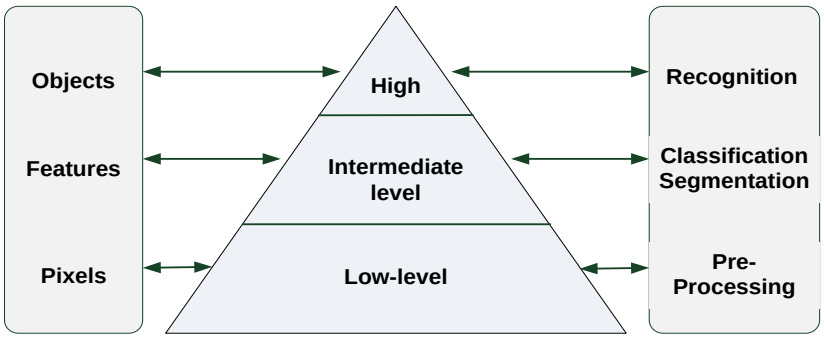


Figure 2.21 Image processing pyramid [71]

This part of this chapter discusses the image processing systems from three different aspects: the IPP and the used algorithms in each part of the pipeline, the existing image processing platform technologies, and finally it presents different examples of imaging processors and architectures.

2.2.1 Image Processing Pipeline

In imaging devices, the produced images pass through a number of enhancement stages to look similar to the scene that we see with our own eyes. These stages are known as IPP. According to Texas Instruments white paper [72], the author defines IPP as the pipeline that "performs the baseline and enhanced image processing,

which takes the raw data produced by a camera sensor and generates the digital image that will then be viewed by the user or undergo further processing before being saved to non-volatile memory". In each stage, specialised algorithms are applied to process the input data. Different algorithms are addressed in the literature for each stage. No algorithm is 100% perfect, and each one has strengths and weaknesses and can be applied for specific conditions. Due to the nature of the pipeline that makes it a high computation power consumer, the pipeline needs to be implemented on a powerful platform or specialised image processors. IPP is located in the base of the image-processing pyramid (see Figure 2.21), which means that a huge load could be removed from the main application if this part is implemented in a different processor.

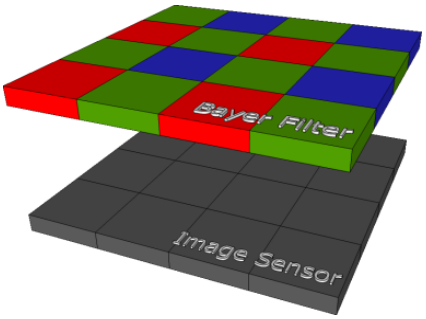


Figure 2.22 Bayer filters and image sensors

Digital images are captured by Complementary Metal-Oxide Semiconductor (CMOS) or Charge-Coupled Device (CCD) image sensors. The sensors are overlaid with an array of light sensitive components that convert the light into electric charges. Although the scene needs to consist of red, green, and blue colours, each component can pass only one colour to reduce the total cost of the sensor, especially in commercial devices [73]. These components are arranged over the sensors in specific form based on different scientific research, such as the Bayer Colour Filter Array (CFA) [74] (see Figure 2.22) and the Red Green Blue Emerald (RGBE) filter announced by Sony Company [75]. The sampled light colours are passed to the IPP to be processed in parallel to maximise the throughput. The IPP sequence of operations is not fixed among different image system manufacturers. Moreover, some IPPs can have more stages to extract more information from the scene or use

novel techniques to process the images. As a conclusion, no specific IPP could be set as a reference for the others. In the literature, a number of IPPs have been discussed or proposed by image specialising companies. Texas Instruments has introduced an IPP known as Davinci IPP, which is widely adopted in many imaging systems (see Figure 2.23) [72]. Each imaging system company tries to develop a special IPP to be used in their products such as the Xylon's IPP from LogicBricks [76]. In [77], the researchers proposed a new IPP to fill the gap between theoretical colour image sciences and practical IPP issues found in digital imaging systems. A typical IPP consists of a number of stages as follows:

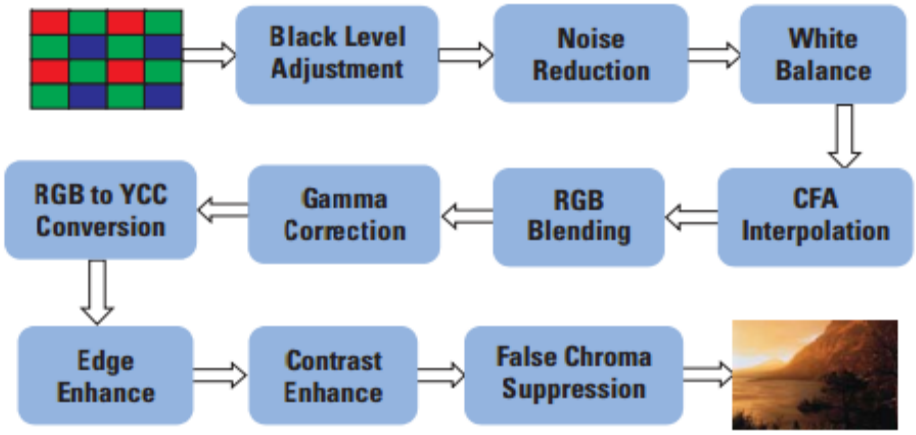


Figure 2.23 Davinci IPP from Texas Instruments [72]

2.2.1.1 Colour Interpolation

In digital cameras, the sensors are covered with a CFA. Each sensor is overlaid with one type of colour filter. This arrangement results in incomplete image samples with two missing colours from each pixel, as shown in the case of the Red-Green-Green-Blue (RGGB) arrangement of the Bayer CFA (see Figure 2.22). Due to the human eye's high sensitivity to green lights, Bayer CFA contains twice as many green (luminance) filter components as either red or blue (chrominance) ones. Based on this structure of CFA, a colour interpolation algorithm is needed to construct the full image and predict the missing colours in each pixel. The interpolation algorithm is known as a demosaicing algorithm. The procedure of any algorithm should first involve extracting individual colours of each pixel and then combining the colours to

form the full image (see Figure 2.24). Demosaicing solutions are divided into two main categories: spatial-domain approaches and frequency domain approaches [78]. Spatial-domain approaches are fast and simple to be implemented but provide a relatively low level of capabilities compared to other methods. They primarily follow spatial and spectral correlation techniques. Many colour interpolation algorithms from spatial domain approaches are presented in the literature. Bilinear interpolation is one of the simplest interpolation algorithms. It interpolates missing colours using symmetric bilinear interpolation from the nearest neighbours of the same colour. It is a resampling that uses the distance-weighted average of the four or two pixel value to estimate the new pixel value [79]. Bicubic is an enhancement of the previous algorithm by involving a 4x4 pixel area to estimate the missing value [80]. Demosaicing algorithms with more complicated approaches have been patented. Some of them, such as Freeman's algorithm [81], use a median filter added to the bilinear interpolation to reduce the noise and artifacts produced by bilinear interpolation. Other algorithms exploit the spatial correlation principle by interpolating along the edges and not across them; these are called edge-based algorithms such as the Adams-Hamilton algorithms proposed in [82], and the asymmetric interpolation scheme using colour discontinuity equalization proposed in [83]. Alternatively, frequency domain approaches exploit the energy spectrum that exists in any image. This spectrum has high frequencies and low frequencies along the vertical and horizontal axes. The frequency-domain approaches exploit the assumption that the human eye is more sensitive to low frequencies than to the high frequency ones. In this context, various interpolation algorithms are discussed in the literature. Some of them, such as [84] and [85], use the Fourier domain for interpolation. Others, such as [86], uses the wavelet domain.

Performance evaluations and comparisons between different demosaicing algorithms have been proposed in a number of scientific papers. A commonly used quantitative measure for the evaluation is Mean Square Error (MSE) or Peak Signal to Noise Ratio (PSNR). In [87], evolutionary work was conducted in more than ten Demosaicing methods to be a reference for any future work in the field.

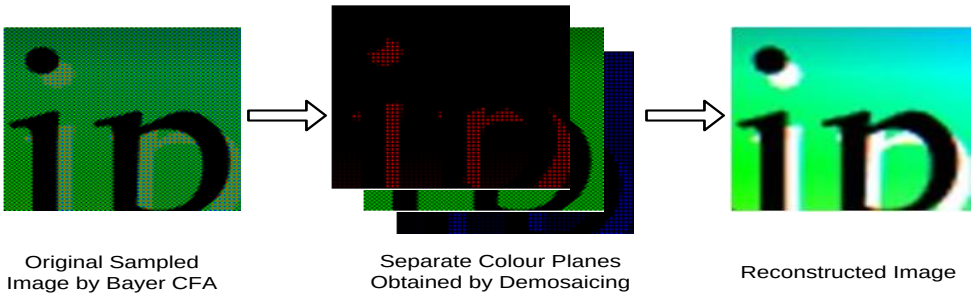


Figure 2.24 Bayer CFA pattern colour interpolation procedure [88]

2.2.1.2 Automatic White Balance (AWB)

AWB or colour constancy is the process of keeping the colour of objects constant automatically under different illumination conditions by calculating a number of parameters from the image data [89]. These parameters are used to change the image pixel values to keep the colour constant. A number of algorithms have been proposed in the literature, such as Gray World Assumption (GWA) [90], Perfect Reflector Assumption (PRA) [91], Retinex theory, standard deviation-weighted gray world, and Gamut mapping method [92]. GWA assumes that the average value in any scene is gray. Therefore, the pixels are rescaled based on that average value. Conversely, PRA assumes that the brightest pixel in the scene is the white colour and rescales the other pixels based on that. Performance evaluation and comparisons were carried out in [93] to compare between different colour constancy methods.

2.2.1.3 Gamma Correction

Gamma correction is the adjustments applied during the display of a digital representation of colour on a screen to compensate for the fact that the e-cathode ray tubes used in monitors generally produce light intensity that is not proportional to the input voltage [72]. Therefore, if the image has to be displayed on a screen, linear Red Green Blue (RGB) components should be converted to a nonlinear signal. Gamma correction controls the overall brightness of the image frames, as they usually appear too dark on screen.

2.2.1.4 Colour Correction

The images captured by digital cameras are affected by many environmental aspects such as illuminations or the object's colour properties. Because of that, we need to

map the data of the captured images to the device colour space. This can be done using a colour compensation chart [72]. Many algorithms have been proposed in the literature dealing with this reproduction IPP stage.

Other stages

The IPP includes other components such as noise reduction, RGB to YCbCr colour space conversion, edge enhancement, and contrast enhancement. Each of these stages enhances the frame in some way and removes any effects that are caused by the surrounding conditions. Additionally, a post-processing compression stage is used to reduce the total image data for storage purposes. A number of compression standards are widely used in the industry, but the primary one is the Joint Photographic Experts Group (JPEG) standard. The JPEG standard is the most commonly used standard for digital cameras due to its simplicity and effectiveness. At this point of processing, the images are ready to be processed by any post-processing component. These components work with features and objects rather than pixels. Literature addresses many topics in this area, including face detection, classification, and image rotation. In terms of memory, all algorithms need a portion of data to be stored in special memory with high-speed access to accelerate data processing. The amount of data depends on the type of operation. For example, the DCT operation requires a large amount of data to be stored, as the operation uses 8x8 pixel blocks, while the thresholding operation needs just one value at a time. This difference in memory requirement between various algorithms must be considered when implementing the pipeline. To get the best out of this pipeline, all the mentioned stages should be implemented in a high-performance platform to fulfil the real-time processing requirements.

2.2.2 Image Processing and Computing Platforms

The ideal platform for image processing should achieve minimum requirements in the following five aspects: performance, programmability (flexibility), parallelism, power efficiency, and cost. A number of platform technologies are used in this field, including hardware-based solutions such ASICs, Application-Specific Standard Products (ASSPs), and FPGAs; and programmable solutions such as DSPs, CPUs,

and media general processors like GPUs. Each of them has its own characterising features. The following paragraphs explain the features of the different platforms.

High performance: The need of a high-performance platform is a must in the case of real-time processing applications such as real-time video encoding and decoding in video applications (e.g. H.264). Not all platforms can fulfil the real-time processing requirement. For example, DSP platforms with 1 GHz frequency cannot perform H.264 HD in real time [94], while they can perform H.264 with low resolution and low frame rate. On the other hand, FPGAs, ASICs, ASSPs, and GPUs can perform it easily due to the amount of parallelism they can provide. CPUs are not in the list of candidates. Mixed platform environments are a good solution for such issues because the high-performance part can handle the most critical tasks. This solution is widely adopted in the industry (e.g. Zynq-7000 SoC).

Programmability: With the increasing number of imaging applications, the need of an adaptive environment to adapt itself to any changes or requirements with low cost becomes important. Moreover, the imaging technologies are constantly evolving, which means the need for flexible platform makes the upgrading process possible and protects them from obsolescence. ASSP and ASIC are the least flexible platforms as they cannot be changed after manufacturing. Alternatively, CPUs, GPUs, and DSPs are flexible platforms, while FPGAs are considered a moderate platform. Any flexibility level is considered in imaging systems.

Parallelism: The computation power of CPUs steadily increased for many decades until they reached the point where is not possible to be inline with Moore's law. The solution at that point is to increase the number of cores. Again, this solution is not a long-term solution. CPUs have limitations in terms of parallelism. On the other hand, FPGAs, ASIC, and GPUs have an amount of parallelism that makes them suitable for image processing applications. In [95], a comparison between CPUs, GPUs, FPGAs, and massively parallel processor arrays is performed and shows that the FPGA can outperform other platforms with slight superiority over GPUs. To utilise parallelism in any platform, a high-speed memory is needed for exchanging data quickly with the processing elements.

Power efficiency: Power consumption recently has become an important design parameter. FPGAs and ASICs are the most power efficient platforms, while GPUs and CPUs are power hungry [96]. In [97], studies showed that FPGAs are power-efficient devices compared to CPUs and GPUs. Table 2.5 shows a power consumption comparison between FPGA, GPU, and CPU implementations for Sum of Squared Differences (SSD) and Sum of Absolute Difference (SAD) [98].

Table 2.5 Power Consumption: SAD & SSD Implementations in Different Platforms [98]

Platform	TDP (W)	SAD Energy (J)	SSD Energy (J)
Core i7 2600K	95	83.77	76.98
NVidia GTX680	195	24.23	24.37
NVidia 8800 Ultra	175	-	286.88
ARM Cortex-7	1.25	53.41	59.24
SIMD Accelerator	1.2	13.01	8.95

Unit cost: ASICs devices require an additional one-time cost known as Non-Recurring Engineering (NRE) cost. It is required to research, develop, design, and test any new product. This cost is relatively high. In the case of high-volume consumer applications that need a big number of units from the same product, the total cost will drop due to the low price of the ASIC devices with NRE. However, in the case of lower-volume consumer applications, the unit cost will be relatively high due to the NRE cost. Conversely, FPGAs have no additional cost on top of the devices.

As a conclusion, FPGAs could be an ideal technology for image processing in the long term. Today, ASICs and GPUs are adopted in a wide range of products in the market, but the lack of flexibility in ASICs and high power consumption in GPU platforms could cause troubles and hinder the rapid progress in the field in the future with the continued evolution of FPGA technologies and the market trend toward embedded systems. Table 2.6 summarises the technologies based on mapping of different characteristics of the technology to particular applications.

Table 2.6 Mapping Technology Characteristics to Application Requirements

	ASIC	**ASSP**	**DSP**	**FPGA**	**CPU**	**GPU**
Customisable	No	No	Yes	Yes (end user)	Yes (end user)	Yes (end user)
Erasable/ Reprogrammable	No	No	Yes	Yes	Yes	Yes
Time-to-Market	Slow	Moderate	Fast	Fast	Fast	Fast
Unit cost	Low (high volume)	Moderate	Moderate	Moderate	Moderate	High
Development cost	High	Moderate	Moderate	Moderate	Low	Low
Field upgradability	No	No	Yes	Yes	Yes	Yes
Power Consumption	Low	Low	Low	Low	High	High
Parallelism	High	High	limited	High	limited	High

2.2.3 Solutions for Image Processing Applications

In the literature, different types of solutions for image processing are addressed. These solutions are divided into different categories based on their architecture or the technology employed, including FPGAs, DSPs, GPUs; reconfigurable architectures, and custom chips such as ASICs. Some of these solutions are implemented for specific functionality, and can be used as accelerators in bigger systems or with GPPs, while the majority of the solutions are used as final products that consist of several functions integrated in a chip (e.g. image processors). In this section, the available image-processing solutions in the literature are discussed.

2.2.3.1 FPGA-Based Implementations

Due to their flexibility compared to other hardware-based solutions, FPGAs are widely adopted in image processing components. Moreover, they can provide the customers with a high-speed environment with impressively low power consumption compared to DSP, CPU, and GPU solutions. This section discusses a number of

FPGA-based solutions including accelerators, image processing IPs, and reconfigurable FPGA-based soft processors.

Accelerators and image processing IPs: There are many successful implementations of IPs or accelerators on FPGAs. In [99], a bilinear interpolation engine has been implemented on Virtex-4 FPGA based on 1920x1080 images. The engine can provide throughput up to 150 MP/s at 150 MHz frequency. The author in [100] proposed a new interpolation method to reduce the computational complexity. The method has been implemented on FPGA. It showed a reduction in required components compared to others. A more generalised architecture for bilinear interpolation has been proposed in [101]. This architecture distributes the image pixels across multiple memories to be accessed in parallel at the time of interpolation. This architecture showed an improvement over the traditional method of implementation. Most of the IPP components have been implemented on FPGA as addressed by different works available in the literature. A static design for the gray edge hypothesis for AWB component has been implemented on FPGA in [102], but no specific information about the performance and resources utilisation was provided. A proposed AWB algorithm has been implemented on FPGA [103]. The engine can process 1920x1080 images at a rate of 75 frames per second, which is the required rate for real-time processing.

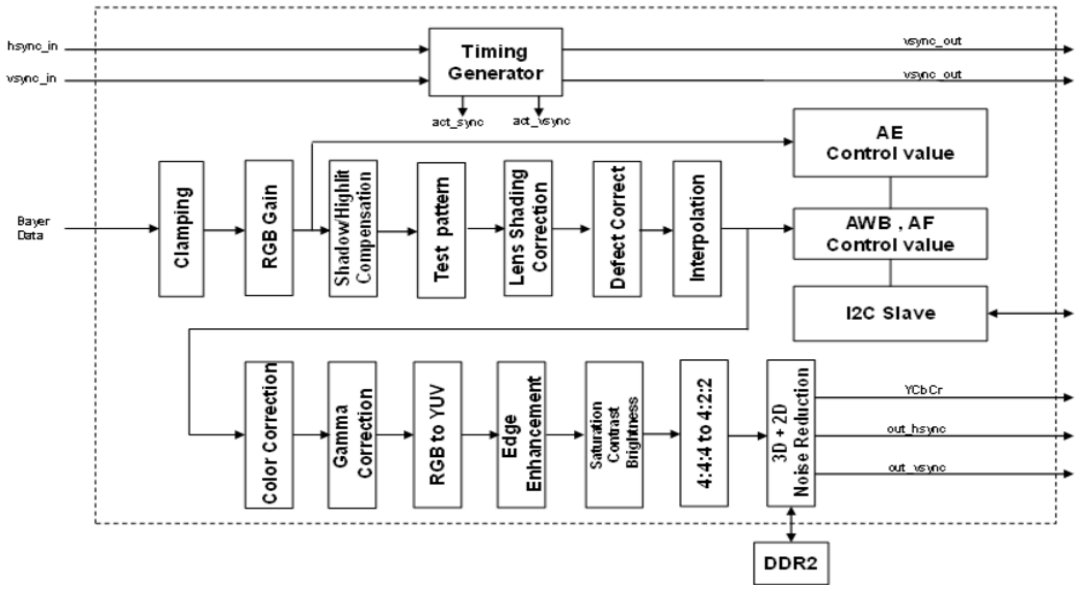


Figure 2.25 Block diagram of FPGAASIC ISP core [104]

More sophisticated implementations are proposed in [104], [105], and [106] in which the whole IPP is implemented as Intellectual Property (IP) to process the raw images and obtain the final output. In [104], IPP has been implemented on FPGA with flexible image size with a complete set of stages including interpolation, defect and colour correction, noise reduction, AWB, edge enhancement, and others. The architecture of the IP based on pipelined stages is shown in Figure 2.25. In [105], Xilinx proposed its own IPP implementation based on Zynq-7000 AP SoC. The implementation works in a configurable manner to adjust the algorithm parameters on the fly, either manually by the user or automatically based on statistical information gathered from the data frames. Individual IPs are connected together to form the full IPP. Another approach of implementation is proposed in [106], in which part of the design is implemented on the FPGA and the remaining part resides in the CPU and GPU platforms. The hybrid implementation uses the FPGA to relieve the GPU and CPU from the burden of computation-intensive tasks. The implementation is capable of processing video streams at resolutions up to 1920x1080 at a 30 frames per second rate.

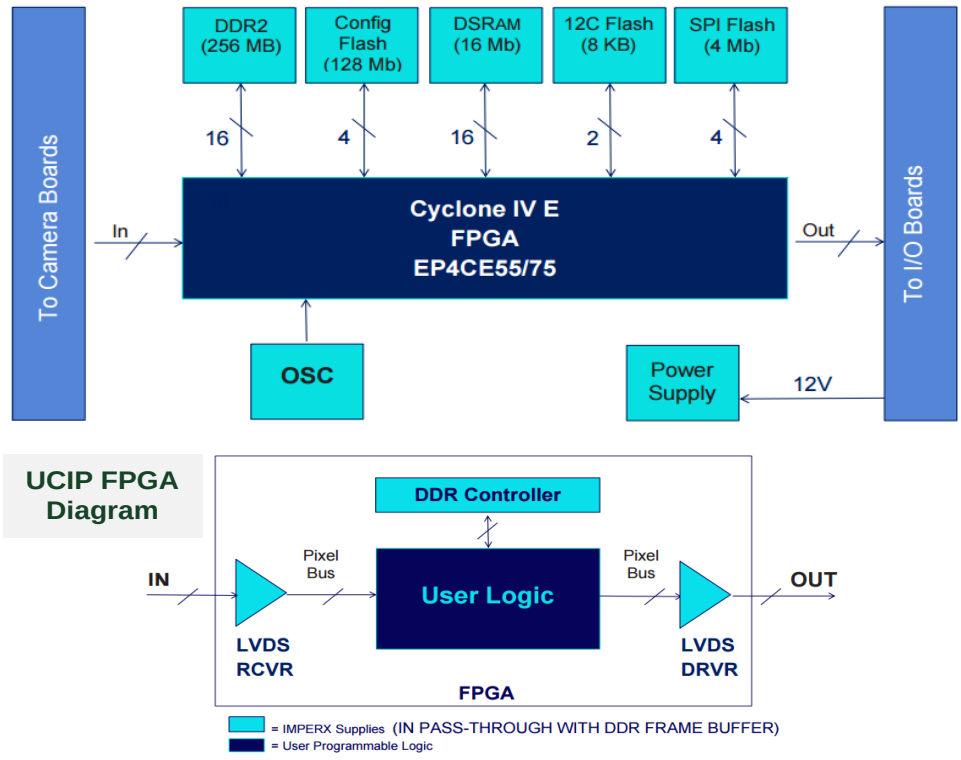


Figure 2.26 IMPERX FPGA-based user customisable image processor [107]

In [108], the AWB stage is split into partitions. The work investigated the best fit environment for each partition based on the total resources utilisation and execution time. In [107], a different idea was proposed. IMPERX Company has introduced a User Customisable Image Processor for cameras in which the FPGA-based processor is fully customisable for signal processing applications. The FPGA has an empty user logic to build the user application (see Figure 2.26). No information was provided about the structure of the customisable processor or the mechanism of building the user application.

Reconfigurable FPGA-based soft processor: A different paradigm for image processing applications is proposed in many research that requires less design effort by reducing the gap between the programmability and design time on one side and the high performance on the other side. Hardware designs are usually mapped to long time-to-market. To reduce this gap, flexible, scalable, and high-performance soft processors are proposed. These soft processors exploit the parallelism in FPGAs to achieve high processing rates while keeping the programmable nature of the processor. In [109], the IPPro soft processor is proposed to fill the aforementioned gap. IPPro is a 16-bit signed fixed-point scalar soft processor based on Reduced Instruction Set Computing (RISC) load/store architecture. The processor is customisable with 8/16/32 bits. The processor uses distributed memory to reduce the hierarchy levels for multi-core architecture and to increase the level of parallelism for both Data Level Parallelism (DLP) and Instruction Level Parallelism (IPL).

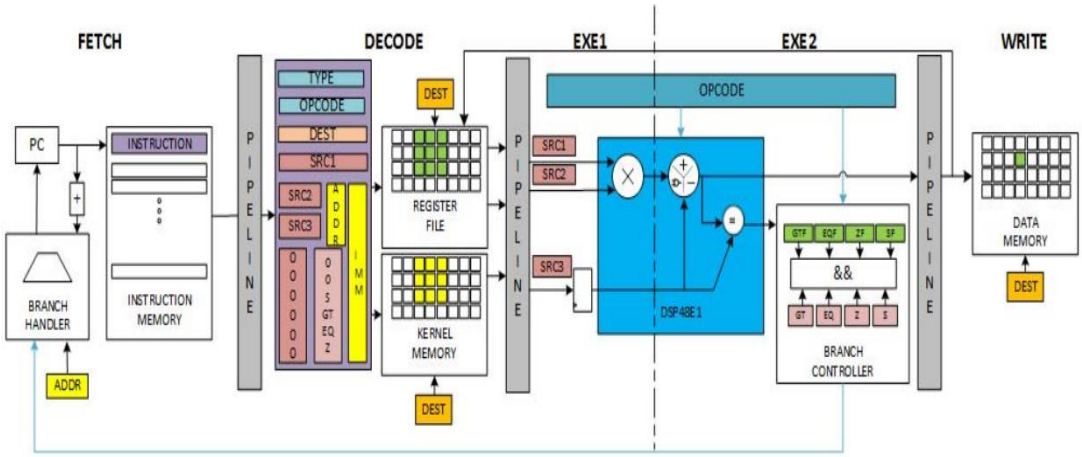


Figure 2.27 IPPro processor datapath [109]

The processor uses load store five-stage balanced pipelined architecture for fixed latency (see Figure 2.27). The IPPro processor supports 53 basic arithmetic and logical operations required by imaging applications. The processor looks similar to the standard processor, but IPPro exploits the DSP48E1 components in the FPGA to implement the instructions. The power of IPPro is in its capability of connecting multi-cores of IPPro together to form a wider processing area with shared memory to increase the performance of the system. The processor has been used to demonstrate the traffic sign detection algorithm based on a 32-core design. The implementation is able to deliver 2.3 fps on images of size 600x400. The resource utilisation of each part of the design is relatively high compared to the expected resource utilisation for accelerator-based design for the same functionality. Moreover, the domain programming language for the IPPro architecture must be developed to automate the process of converting the high-level algorithm into code realised by the IPPro architecture on the FPGA. In [110] and [111], a similar idea is implemented using soft vector processors by exploiting DLP to execute the tasks faster. Similarly, iDEA [112] is a nine-stage pipelined DSP-based soft processor. The iDEA processor runs at 407 MHz compared to 210 MHz for the Microblaze. Moreover, it needs less logic resources compared to Microblaze.

2.2.3.2 Reconfigurable Architecture-Based Implementations

Due to the needs of more flexible and high-performance architectures, a new category of programmable processor architectures, termed coarse-grained reconfigurable architecture (CGRA), has emerged for DSP, high-performance, and area efficient applications. Coarse-grained architecture refers to the array of a large number of reconfigurable Function Units (FUs) interconnected by a special network style [113]. In another words, the architecture is characterised by wider arithmetic logic-oriented datapaths, made up of hardwired arithmetic and logic components such as multipliers, adders, and ALUs [114]. On the other hand, fine-grained architecture refers to the use of bit-wide processing elements such as CLB components in the FPGA [113]. The coarse-grained architecture shares the same concept of reconfigurable soft-processors in FPGAs discussed in section 2.2.3.1. The

FUs can be reconfigured to implement different algorithms with low configuration information compared to FPGAs, as they offer software-like programmability similar to GPP architectures. Moreover, the architecture is more area efficient and power efficient compared to FPGAs but has a lower level of flexibility. A typical CGRA consists of FUs arranged in a 2-D array with routing resources (see Figure 2.28). In the literature, a number of CGRAs are proposed to target image-processing and high-performance applications.

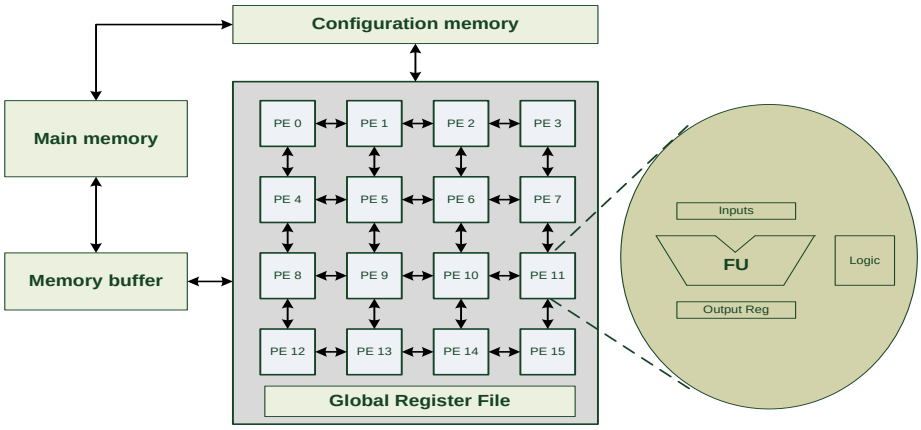


Figure 2.28 General CGRA architecture

In [115], the author proposed a Coarse-Grained Reconfigurable Instruction Set Processor (CRISP) for image processing applications. The processing elements in this architecture are not identical to each other; each of them is specialised in a specific image processing functionality. Due to the aforementioned characteristic, the architecture looks different from other CGRAs and is more ASIC-like. CRISP consists of reconfigurable interconnects, context registers, main controller, and the reconfigurable specialised processing elements including down-sampler, colour interpolation, pixel-based processing element, and the accumulator and multiplication processing element. The author implemented image processing tasks using the proposed architecture and compared their performance with other similar implementations using different architectures. In [116], a Multimedia Oriented Reconfigurable Array (MORA) coarse-grained processor was proposed for accelerating media processing applications. It is a 2-D mesh-based CGRA consisting of an array of cells arranged in four 4x4 quadrants (see Figure 2.29). It is less

complex and more powerful compared to similar platforms. The single reconfigurable cell consists of a controller, 8-bit processing element, and 256x8 dual port data memory. The processors are programmed using a specially developed MORA assembly language.

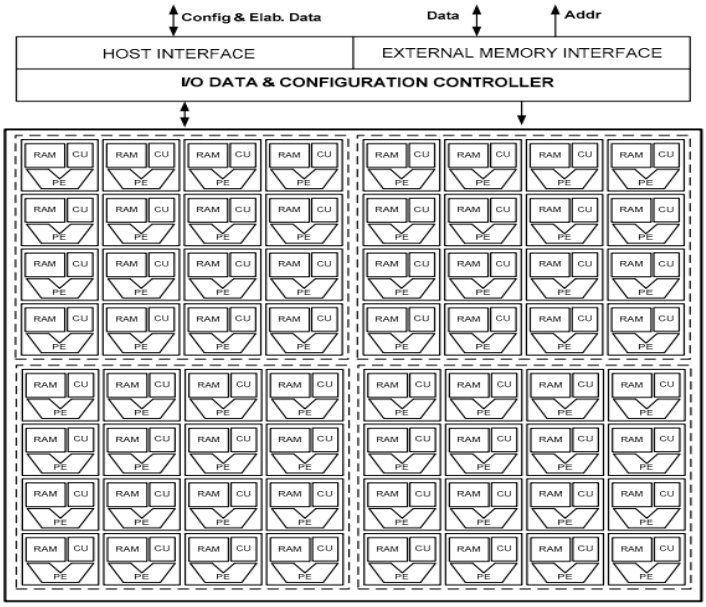


Figure 2.29 2-D array of MORA processor [116]

The paper demonstrated image processing functions using the proposed architecture based on the 64 reconfigurable cells with parallel processing. The implementation can process the 4x4 H.264 with a throughput up to 41.67 MOPS when eight blocks are processed in parallel. In [117], the work proposed the Reconfigurable Instruction Cell Array (RICA) coarse-grained processor architecture (see Figure 2.30). The architecture adopts a different concept from other CGRAs. Instead of using homogeneous processing elements like most CGRAs, RICA has its own heterogeneous coarse-grained hardware modules termed Instruction Cells (ICs) that are interconnected through an island-style mesh. Each IC can work independently and concurrently and can be configured to do a limited type of operations (e.g. ADD IC is used only for addition and subtraction operations and so on for MUL, SHIFT, and REG ICs) to minimize area. The ICs and interconnects can be dynamically reconfigured on every clock cycle, enabling the mapping of data-paths of dependent and independent instructions. The ICs can be configured using C coding, which is

converted to intermediate assembly file and then to RICA-based architecture. In [118], a dual-core reconfigurable demosaicing engine based on RICA architecture has been demonstrated. Freeman's demosaicing algorithm has different odd/even equations. The paper discussed the concept of dividing the algorithm between two RICA cores working in parallel. The architecture provides throughput up to 241 MP/s.

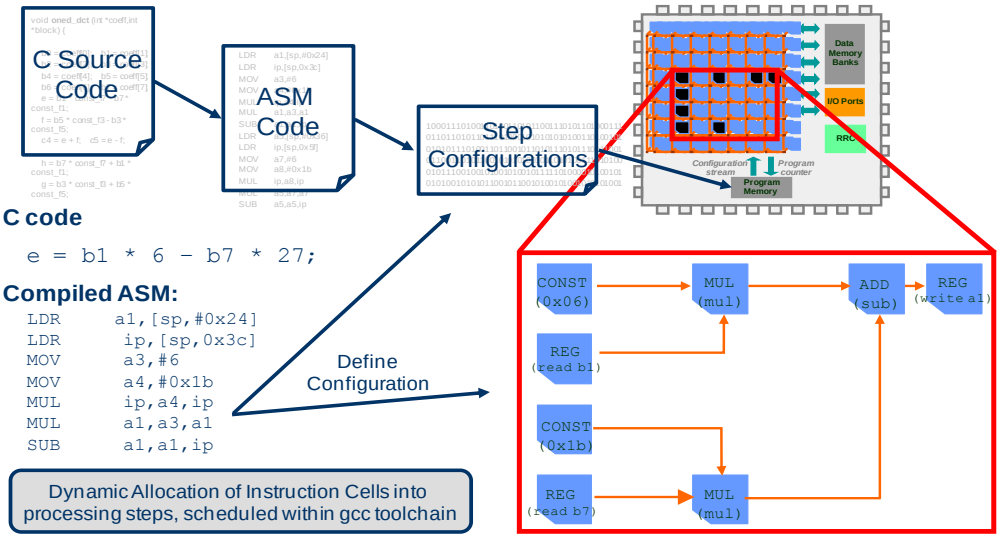


Figure 2.30 RICA block diagram and paradigm [88]

Other CGRAs are discussed in literature. One of them is MorphoSys [119], which combines both coarse-grained and fine-grained techniques in the same architecture for more flexibility. The ADRES architecture proposed in [120] is a combination of the VLIW processor and coarse-grained reconfigurable matrix. FUs in the coarse-grained part support 32-bit wide data-path operands.

2.3 Summary

The first part of the chapter has introduced the architecture of the latest Xilinx SRAM-FPGA with a brief history of the technology. An introduction to the significant reconfigurable resources was presented. This chapter also provided the design flow for different FPGA tools from various FPGA manufacturers. The architecture of the latest Xilinx Zynq-7000 AP SoC and its features have been discussed in detail. Moreover, DPR features and its flow are addressed in detail. To

show the importance of the DPR, the deployments of DPR in high-performance and fault-tolerant systems were also presented. In addition, the chapter discussed relevant research works related to the DPR and its use for enhancing performance, reducing power consumption, and utilising resources. It also discusses the methods of increasing the bandwidth when using DPR to configure bitstreams. This part also discusses the reliability issues in the FPGA and the ways to mitigate soft errors using the embedded features in FPGA or using additional techniques.

The chapter has discussed three main points: the basic image-processing pipeline, existing image processing platform technologies, and different image processors and architectures.

The second part introduced some of the fundamentals of digital image processing systems and discussed it from different aspects. It examined solutions to fulfil the requirements of real-time image processing applications. The part has presented typical image-processing pipelines based on different architectures. It has discussed the stages of IPP, associated algorithms for each stage, and the differences between the stages based on the information available in the literature. It also presented the image processing platform technologies including FPGAs, DSP, ASICs, GPUs, and others. The section has investigations to best platforms for image processing applications based on the following features: performance, programmability, parallelism capabilities, unit cost, and power efficiency. The final section has presented different solutions in the literature based on different platforms. Currently, the most used solution in the market is the custom chip solution due to the high performance and power efficiency it can provide; however, due to the lack of flexibility for these architectures, manufacturers have begun looking for different solutions. FPGAs and CGRAs are very good alternatives due to their high flexibility, and area and power efficiency. Alternatively, DSP solutions have limited ILP, which leads to a lower total throughput. The chapter has presented different architectures proposed in the literature and their features. Moreover, it has discussed implementations of IPP stages using the proposed architectures.

Chapter 3 : Efficient Implementation of the Adams-Hamilton's Demosaicing Algorithm on FPGAs

Most of the commercial portable devices that are used to capture digital images such as digital cameras or mobile phones, employ an array of filters on top of their image sensors. In the case of the RGB image format, each pixel value is produced through sensor overlaid with one type of colour filter to reduce the overall cost of the sensors and devices. These types of cameras are known as single-CCD cameras. Conversely, three-CCD cameras use three filters on top of each sensor to produce all three colours. The filter arrangement in single-CCD results in incomplete image samples with two missing colours from each pixel. Using CFA [74], the sensors produce a two-dimensional array of pixels, each representing a single colour: red, green or blue. Many types of colour filter arrays are used in digital capturing devices; the most common type is the Bayer CFA. Bayer CFA has many colour arrangements. The RGGB arrangement of the Bayer CFA is one of these arrangements (see Figure 3.1). In this arrangement, the filters are 50% green, 25% red, and 25% blue. Half of the sensors are coupled with filters for the green colour, the colour for which the human eye is more sensitive.

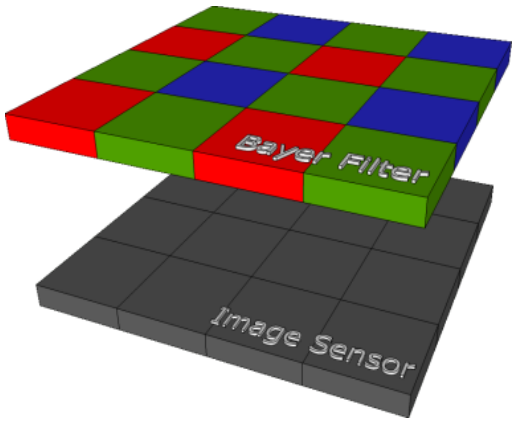


Figure 3.1 Bayer filters and image sensors

Figure 3.2 shows the difference between single-CCD and three-CCD cameras in terms of producing the RGB planes. The images captured by three-CDD cameras have the original content because all the three colours are produced at capture time. To construct a full RGB image in single-CCD cameras, demosaicing algorithms are used to interpolate missing red, green, and blue sub-pixels from the surrounding pixels. Different algorithms have different image qualities and vary in terms of computation demands, but in general, all algorithms require intensive computation. In section 2.2.1.1, many interpolation methods have been discussed in general. Many evaluation and comparative studies such as [121-123] have been carried out for different demosaicing algorithms to find out the best-fit method for most applications. Using FPGAs, demosaicing algorithms can be accelerated by performing some acceleration tasks in hardware to meet the high computation requirements to fit in real-time applications. In this chapter, an efficient implementation of Adams-Hamilton's demosaicing algorithm on FPGA is proposed to provide high throughput to suite real-time image processing applications. The design is implemented using two approaches. Moreover, an evaluation study between these approaches is carried out on this work to find the best-fit approach.

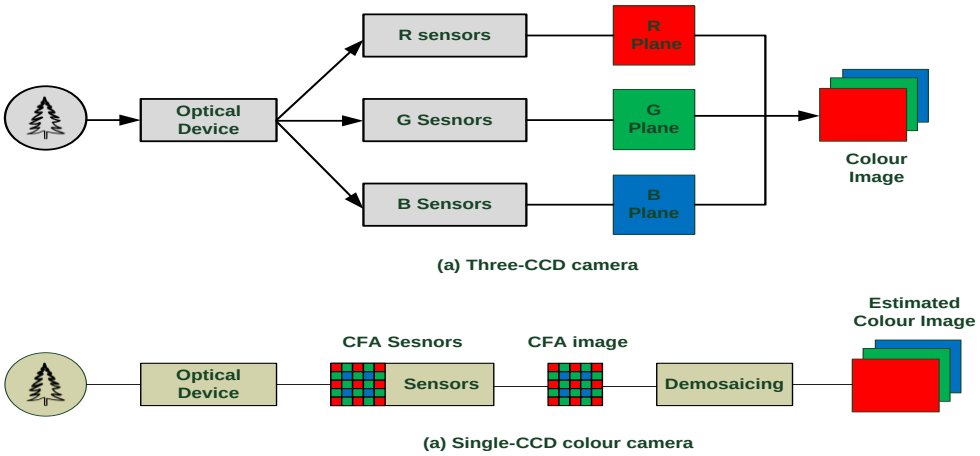


Figure 3.2 Colour image acquisition outline

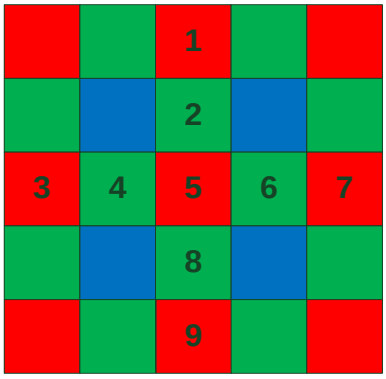
3.1 Adams-Hamilton Demosaicing method

Based on the evaluation and comparison studies in [121-123], Adams-Hamilton's Demosaicing is chosen for the targeted application due its overall good performance

and low implementation and computation complexity [121]. Moreover, PSNR and MSE measurements have been used in these studies to estimate the quality of the interpolated images. The Adams-Hamilton algorithm showed very good results compared to other methods. The Adams-Hamilton's algorithm is considered one of the edge-based algorithms that exploits the spatial correlation principle by interpolating along the edges and not across them. This technique reduces colour artifacts and zipper effects to the regions with edges, unlike the other types of algorithms that disregard directional information. Moreover, averaging the pixels across an edge will decrease the sharpness at edges.

The algorithm is divided into two steps. First, the green colour plane is interpolated; then, the red and blue planes are interpolated. G missing pixels can be interpolated vertically, horizontally, or using the two directions based on specific classifiers (composed of Laplacian second-order terms for the chroma data and gradients for the green data) to choose the interpolation direction (see Figure 3.3). In the case of G pixels in blue positions, the same equations are used as red positions replaced with blue. G pixel estimation uses a window of 5x5 pixels. Once the G colour plane is interpolated, the algorithm starts interpolating the red and blue colours. In this colour estimation, a window of 3x3 is needed (see Figure 3.4). This step is categorised into three different cases:

1. **N**earest neighbours to (R or B) are in the same column.
2. **N**earest neighbours to (R or B) are in the same row.
3. **N**earest neighbours to (R or B) are at the four corners.



1. Calculate horizontal gradient
$\Delta H = |G4 - G6| + |R5 - R3 + R5 - R7|$
2. Calculate vertical gradient
$\Delta V = |G2 - G8| + |R5 - R1 + R5 - R9|$
3. If $\Delta H > \Delta V$,
$G5 = (G2 + G8)/2 + (R5 - R1 + R5 - R9)/4$
Else if $\Delta H < \Delta V$,
$G5 = (G4 + G6)/2 + (R5 - R3 + R5 - R7)/4$
Else
$G5 = (G2 + G8 + G4 + G6)/4$
$+ (R5 - R1 + R5 - R9 + R5 - R3 + R5 - R7)/8$

Figure 3.3 G pixel estimation at pixel 5 using Adams-Hamilton's method

In cases one and two, the missing Rs or Bs are in G locations (see Figure 3.4a). To estimate the missing colours, blue and red classifiers are used. These classifiers appear in equation 3.1 if the nearest neighbour in the same column (Blue Channel) and equation 3.2 if the nearest neighbour in the same row (Blue Channel). Equations 3.3 and 3.4 show the Red Channel classifiers. Equation 3.3 shows the classifier if the nearest neighbour in the same column and 3.4 shows is if the nearest neighbour in the same row.

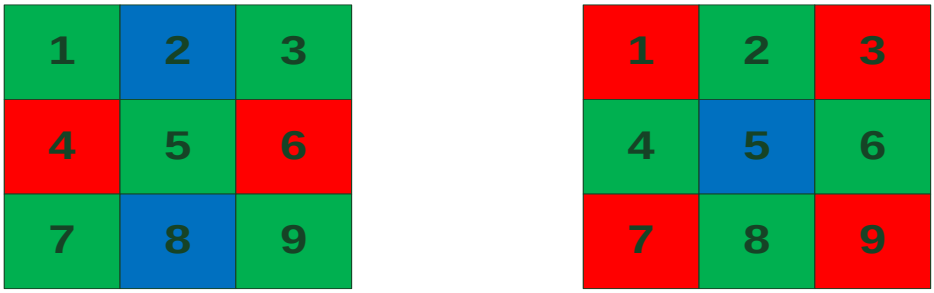


Figure 3.4 (R or B) pixel estimation (a) case one and two (b) case three

$$B5= (B2+B8)/2+ (-G2+2G5-G8)/2 \quad (3.1)$$

$$B5= (B4+B6)/2+ (-G4+2G5-G6)/2 \quad (3.2)$$

$$R5= (R2+R8)/2+ (-G2+2G5-G8)/2 \quad (3.3)$$

$$R5= (R4+R6)/2+ (-G4+2G5-G6)/2 \quad (3.4)$$

Case 3 is used for the missing R pixel part in B locations as the case depicted in Figure 3.4b or the missing B pixel part in R locations. The nearest neighbours are at the corners of the 3x3 window. Classifiers composed of Laplacian second-order terms for the green data and gradients for the chroma data are used. These classifiers sense the high spatial frequency information present in the pixel neighbourhood in the Negative Diagonal (DN) and Positive Diagonal (DP) directions as shown in equations 3.3 and 3.4 for the case of the missing R pixel part in B locations. Based on the classifiers, the direction of interpolation is determined as shown in equations 3.5, 3.6, and 3.7. Equations 3.10 to 3.14 show the case of missing B in R location.

Red Channel case, missing R in B location:

$$DN = |R1 - R9| + |G5 - G1 + G5 - G9| \quad (3.5)$$

$$DP = |R3 - R7| + |G5 - G3 + G5 - G7| \quad (3.6)$$

If DN > DP,

$$R5 = (R3 + R7)/2 + (-G3 + 2G5 - G7)/4 \quad (3.7)$$

Else if DN < DP,

$$R5 = (R1 + R9)/2 + (-G1 + 2G5 - G9)/4 \quad (3.8)$$

Else

$$R5 = (R1 + R3 + R7 + R9)/4 + (-G1 - G3 + 4G5 - G7 - G9)/8 \quad (3.9)$$

Blue Channel case, missing B in R location:

$$DN = |B1 - B9| + |G5 - G1 + G5 - G9| \quad (3.10)$$

$$DP = |B3 - B7| + |G5 - G3 + G5 - G7| \quad (3.11)$$

If DN > DP,

$$B5 = (B3 + B7)/2 + (-G3 + 2G5 - G7)/4 \quad (3.12)$$

Else if DN < DP,

$$B5 = (B1 + B9)/2 + (-G1 + 2G5 - G9)/4 \quad (3.13)$$

Else

$$B5 = (B1 + B3 + B7 + B9)/4 + (-G1 - G3 + 4G5 - G7 - G9)/8 \quad (3.14)$$

In addition to the Adams-Hamilton method, an optional 3x3 median filter has been used to reduce the artificial noise by removing any sudden jumps in the hue. The median filter is working in a window-based format. The window slides one-step each clock cycle to load new values. The median filter replaces the middle pixel in the window with the middle value after all the entries of the window are sorted numerically [124]. Algorithm 3.1 shows the pseudo code for the 2-D true median filter.

```
allocate outputPixelValue[image width][image height]
allocate window[window width * window height]
edgex := (window width / 2)    rounded down
edgey := (window height / 2)    rounded down
for x from edgex to image width - edgex
   for y from edgey to image height - edgey
      i = 0
      for fx from 0 to window width
         for fy from 0 to window height
            window[i] := inputPixelValue[x + fx - edgex][y + fy - edgey]
            i := i + 1
      sort entries in window[] until reach position window width * window height / 2
      Stop sorting
      outputPixelValue[x][y] := window[window width * window height / 2]
```

Algorithm 3.1 : Two-dimensional median filter pseudo code [125]

3.2 Hardware Implementation

The algorithm has been implemented using two methods in this chapter: RTL implementation and HLS implementation using the Xilinx HLS tool. VHDL has been used for RTL implementation, while C code has been used in the HLS implementation. The two methods are fully demonstrated to compare them in terms of performance and resources utilization. The implementations are demonstrated on Virtex-6 and Zynq-7000 AP SoC. In the case of Virtex-6, a Microblaze soft processor is used to control the design, while Cortex ARM hardwired processor is employed in the case of Zynq implementation.

3.2.1 RTL Implementation

The RTL implementation of Adams-Hamilton demosaicing is illustrated in Figure 3.5. The implementation is based on the RGGB Bayer CFA pattern. As the images are processed line-by-line from left to right, a set of data buffers is employed as storage for each line. The size of the data buffer depends on the width of the image. The implemented system is designed to process images of size 1920x1080 pixels.

Shift windows of sizes 5x5 and 3x3 are used in the G interpolation and R&B colour interpolation stages, respectively.

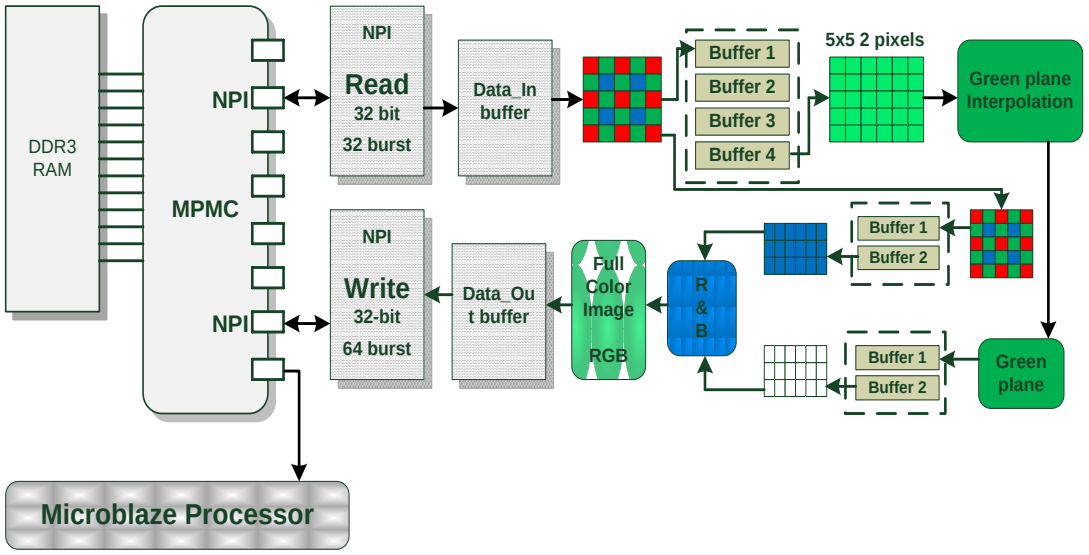


Figure 3.5 Adams-Hamilton RTL implementation block diagram

The system is designed to process two pixels each clock cycle using an efficient pipelining scheme. The data input is designed to accept 2 bytes (pixels), and the processes are parallelized in the data path. This technique will double the performance but will increase the required resources. The maximum number of pixels to be processed in one clock cycle depends on the memory interface, as each byte in the input produces three bytes at the output side. The design is memory-to-memory based. The image data are stored in the memory prior to the processing and should be saved again later in the memory to be ready for further processing.

Data Buffering

The size of each FIFO is 1914 bytes. This number is based on the number 1920, the width of the processed images. The remaining six bytes are located in the shift registers where the window is located. BRAMs are used to construct FIFOs for each line in the shift window. For the green interpolation stage (see Figure 3.6), the four FIFOs contain the image lines currently under processing and the shown shift registers present the actual window.

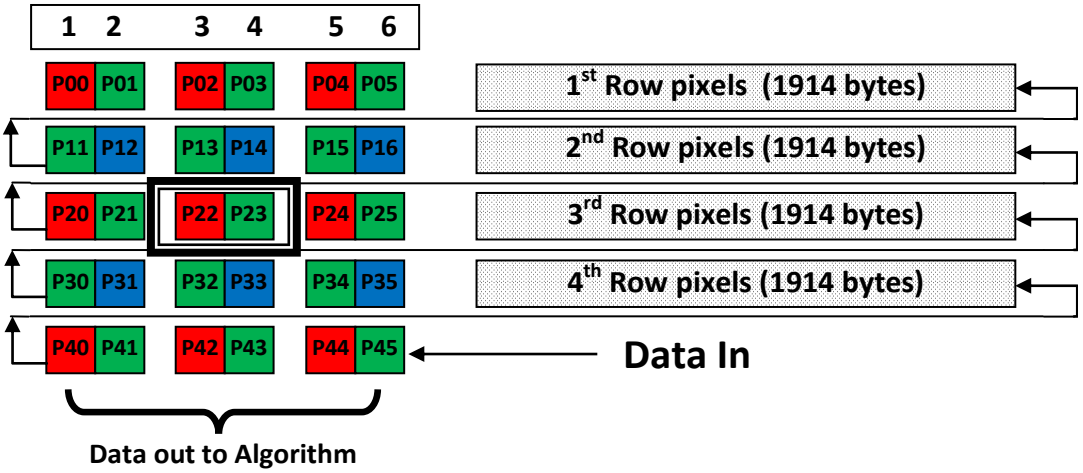


Figure 3.6 Buffers of G interpolation stage

The red and blue interpolation uses the same structure but employs two FIFOs instead of four. P22 and P23 are the locations of the processed pixels at each clock cycle. In each clock cycle, a shift operation occurs by shifting the bytes in registers 1 and 2 to the next row buffer; bytes in registers 3 and 4 to registers 1 and 2; and, finally, the most significant two bytes in the same row buffer to registers 5 and 6. Inside the buffer, if the data of the image are still coming, the write address and read addresses are incremented when a shift operation occurs; otherwise, the read address is only incremented until the two addresses are equal, which signifies the end of the image data. The number of FIFOs is equal to N-1, where N is the number of lines (or rows) to be saved in the memory to build up the NxN window. The data of the last row of the current window are fed directly into the window registers to optimise the RTL implementation. Earlier implementations were consuming more memory resources by using N buffer lines to build up the NxN window. This method of data buffering is very common and is widely adopted in image processing implementations for situations in which the memory resources are a bottleneck, as our case here with FPGAs. In [126-128], the authors have used the same line buffer architecture to process the data in the register and to reduce the use of memory resources.

Image Interpolation

In each clock cycle, the content of registers 1 to 6 is sent to the equivalent hardware design of the equations mentioned in section 3.1. The hardware design defines the selected method with its two sets of equations for the two stages: the G interpolation stage and the R & B interpolation stage. The implementation is divided into two parts based on the type of equations:

1. Even rows: In these rows, the received six bytes are arranged in the form RGRGRG. The algorithm interpolates the missing colours in the middle locations where the pixels R and G are located (see Figure 3.6). For the G interpolation stage, only the first Green pixel is interpolated., The the second G pixel is contained in the original data. The G interpolation uses the equations in Figure 4.3. However, in R & B interpolation stage, the B part is unknown in the first pixel, and the (B and R) parts are unknown in the second pixel.
2. Odd rows: In these rows, the received six bytes are arranged in the form GBGBGB. The design in these rows acts in the same manner as the even rows but different colours are interpolated due to the change of the location of each colour. In the G stage, the green colour part is missing in the second pixels in the odd rows. Conversely, the red and blue colour parts are missing in the first pixel, while the red colour part is missing in the second pixel for R & B interpolation stage.

The equations contain addition, subtraction, and division by two, four, and eight operations. No floating points are needed to implement the algorithm. The division operations are implemented using shift operations only. Due to the use of shift operations, inaccurate results are extracted with margin of error -/+ N, where N is the divisor. Additional logic (modulus logic) has been added to overcome the error margin.

Although it needs more time to calculate the median value, a true median filter has been implemented rather than a pseudo median filter [129] for obtaining a more accurate result. The pseudo median uses only two stages: 1) get the median value for each column, and 2) get the median value for the median values in step one. It is a simple but inaccurate method. The true median filter needs more steps to obtain the median value based on Algorithm 3.1. To sustain the high performance with the true

median filter, intermediate registers are deployed to increase the frequency of the implementation by increasing the pipeline stages. This deployment will increase the resources needed for the entire system.

Memory Interface

In Virtex-6-based implementation, the Xilinx Native Port Interface (NPI) [130] is used to connect the system to the DDR memory through the Xilinx Multi Port Memory Controller (MPMC) [130]. Two ports of the MPMC are dedicated, one for reading data and the other for writing. Due to the difference in data size between the NPI part and the demosaicing engine, two blocks in the middle, named Data_In and Data_out, are implemented to convert the size of data from the NPI form to the system form, and vice versa. A Microblaze processor is used to control data transfer from to the memory side and to control the number of required bursts.

3.2.2 HLS Implementation

The second implementation of the algorithm is based on the Vivado HLS tool. Instead of describing the algorithm from the RTL perspective by defining the datapath and logic using VHDL, high-level equivalent code can be synthesized directly by the tool to represent the algorithm functionality. The high-level code is then transformed to RTL implementation. This extra level of abstraction hides the complexity of the lower-level implementation step.

For simplicity, the Vivado HLS tool accepts codes written using C/C++ or SystemC languages. The desired algorithm can be divided into functions. Each function is converted to the RTL module, where all functions define the hierarchy in RTL. The hierarchy and datapath of Adams-Hamilton's algorithm with median filter are illustrated in Figure 3.7 for HLS implementation. The wrapper function defines the top-level I/O ports and default protocols. To generate an optimised RTL code, the tool uses a number of optimisation libraries, such as function and dataflow pipelining, resources limitation, and loop unrolling, to map the code efficiently to the FPGA gate level. Moreover, a number of ready memory structures, including line

buffer and window structures, are available for use in most of the HLS applications to reduce the coding effort. The HLS-based implementation is able to process one or two pixels per clock cycles based on the input stream width and the width of data in each function.

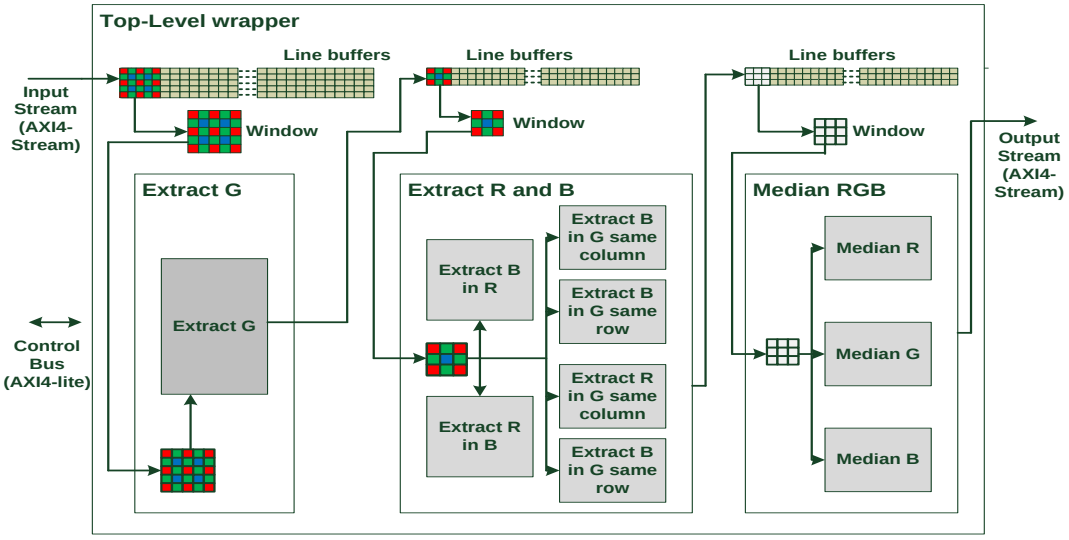


Figure 3.7 HLS implementation block diagram

Memory Structures: Two main memory structures have been used in this implementation: memory window and line buffer.

Memory window is widely used in imaging applications that need more than one image line to compute pixel P. P is the centered pixel of NxN window pixels, where N is an odd number. In HLS, the window is defined as a 2-D array, where all elements of the array are available simultaneously. This can be achieved by partitioning the array into individual elements to force the tool to implement the array as registers rather than block RAM, as shown in Algorithm 3.2, where a 3x3 window of RGB pixels is defined.

```
ap_window(){                        /*  force array partitioning */
#pragma AP ARRAY_PARTITION variable=M dim=0 complete
};

/* Define window type, NxN and elements */
typedef ap_window<RGB,3,3> WINDOW;

/* Define variable of RGB Window */
WINDOW buff_window;
```

Algorithm 3.2 Window-based structure partitioning and definition

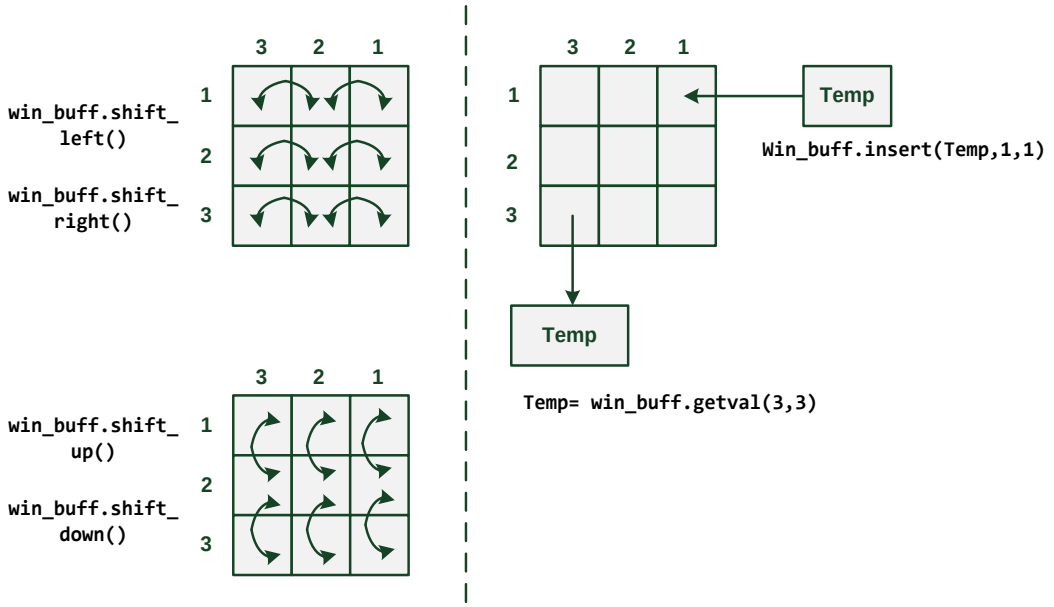


Figure 3.8 Window structure functions

The memory window structure has a number of related functions to ease the use of the structure while coding; these include **`shift_right(),shift_left(),shift_up(), shift_down(),insert(value,row,col), and getval(RowIndex, ColIndex)`** (see Figure 3.8).

Line buffer has a similar structure to memory window but with different functionality. Line buffer is in the form of NxM, where M is the image width, -line- and N is number of lines to be held by the buffer. Line buffer is used to hold the image lines that are involved in the computation. In HLS, line buffers are implemented using block RAMs to avoid any communication latency and to benefit from the dual access read/write- feature of the BRAM. Similar to memory window, the line buffer structure has a number of functions to ease coding in HLS, including **`shift_up`**`(col),` **`shift_down`**`(col),` **`insert_top`**`(value, col),` **`insert_bottom`**`(value, col), and` **`getval`**`(RowIndex, ColIndex).` The difference here is that the shifting functions are only for one column, not for the whole columns. Combining the window and line buffer structures together enables the user to activate the algorithm computation kernel (see figure 3.7).

Performance optimisation: The aforementioned memory structures are used to enhance the performance of the implementation. Moreover, loop unrolling and pipelined functions and pipelined dataflows are different ways to enhance the performance. In normal loops, all operations in the loop are implemented using the same hardware resources for the iteration in the loop [6], which means that the loop needs at least N clock cycles to execute the loop operations. On the other hand, unrolling the loop creates multiple independent operations rather than a single collection to execute the loop in just one clock cycle. This practice will increase the resources needed in the implementation. The most powerful feature that enhances the performance is dataflow pipelining. This feature allows the sequential functions, loop, or dataflow to operate concurrently at the RTL (e.g loop pipelining), as shown in Figure 3.9. HLS adds channels or registers between the operations to hold the intermediate data that flow between the functions. Again, this feature will affect the resources of the implementation. The following command is used to apply this feature in each function and in the main function.

```
#pragma AP PIPELINE II = 1
```

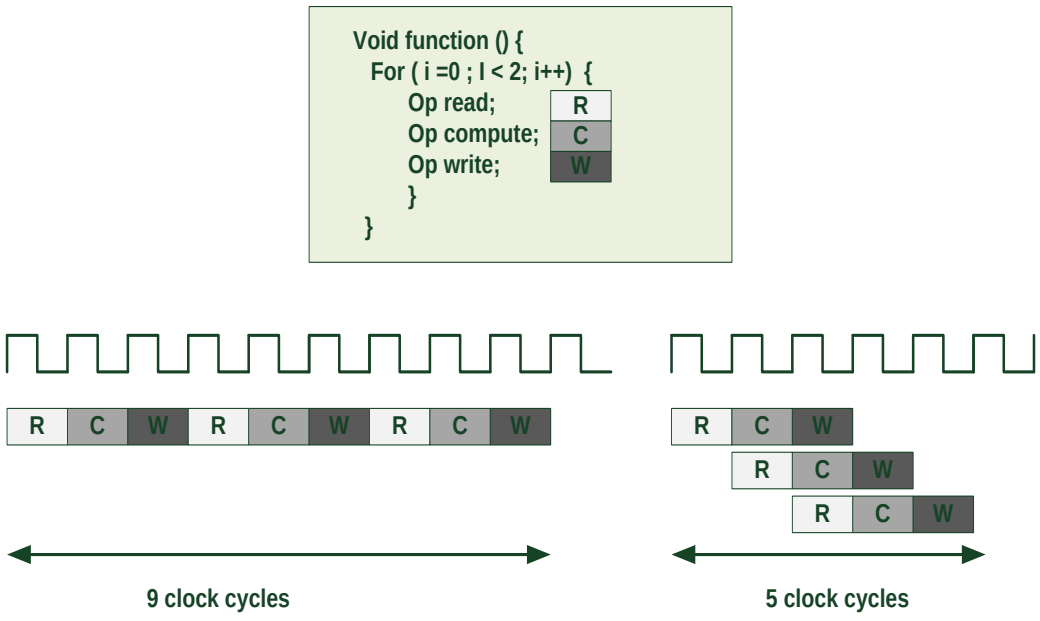


Figure 3.9 Pipeline instructions in Loop using HLS PIPELINE II=1 command [6]

Interfacing: I/O ports consist of two types, default ports and user-defined ports. Default ports include clock, reset, and enable ports. They are automatically defined by the HLS tool when the code is synthesised. Alternatively, user-defined ports are defined based on the user requirements. These ports are defined as function parameters in the main function. In this implementation, two types of user-defined ports and bus Interfaces have been defined: AXI4-lite slave and AXI4-stream bus interfaces. AXI4-lite is used to allow implementation to be controlled by CPU. Conversely, the AXI4-stream is used to handle a streaming data from external memory or adjacent component. The Vivado HLS tool takes care of all the needed actions to define the buses to keep the user concentrating on the design functionality. Algorithm 3.3 defines the algorithm ports and the type of each port.

```
void CFA_32(AXI_PIXEL inter_pix[MAX_HEIGHT]z[MAX_WIDTH],AXI_PIXEL
out_pix[MAX_HEIGHT][MAX_WIDTH], int rows, int cols , int add_filter,
int phase)

  AP_BUS_AXI_STREAMD(inter_pix,INPUT_STREAM);
  AP_BUS_AXI_STREAMD(out_pix,OUTPUT_STREAM);
  AP_INTERFACE(rows,ap_none);
  AP_INTERFACE(cols,ap_none);
  AP_BUS_AXI4_LITE(rows, CONTROL_BUS);
  AP_BUS_AXI4_LITE(cols, CONTROL_BUS);
  AP_CONTROL_BUS_AXI(CONTROL_BUS);

  AP_INTERFACE(add_filter,ap_none);
  AP_BUS_AXI4_LITE(add_filter, CONTROL_BUS);

  AP_INTERFACE(phase,ap_none);
  AP_BUS_AXI4_LITE(phase, CONTROL_BUS);
```

Algorithm 3.3 Bus interfaces definition

3.3 Experimental Results

The above implementations were tested on two platforms: Xilinx ML605 boards with a Virtex-6 XCE6VLX240T FPGA chip for the RTL-based implementation and Zynq 7000 AP SoC for the HLS implementation. Both implementations are able to process two pixels per clock cycles. For testing, the data is taken directly from camera sensors. The data are either kept in DDR memory for later access or processed immediately by the interpolation algorithm (see Figure 3.10 and Figure 3.11). The Vita-2000 camera is used in the testing platform. The camera is connected to the system using an Avent (FPGA Mezzanine Card) FMC adapter through an FMC connecter.

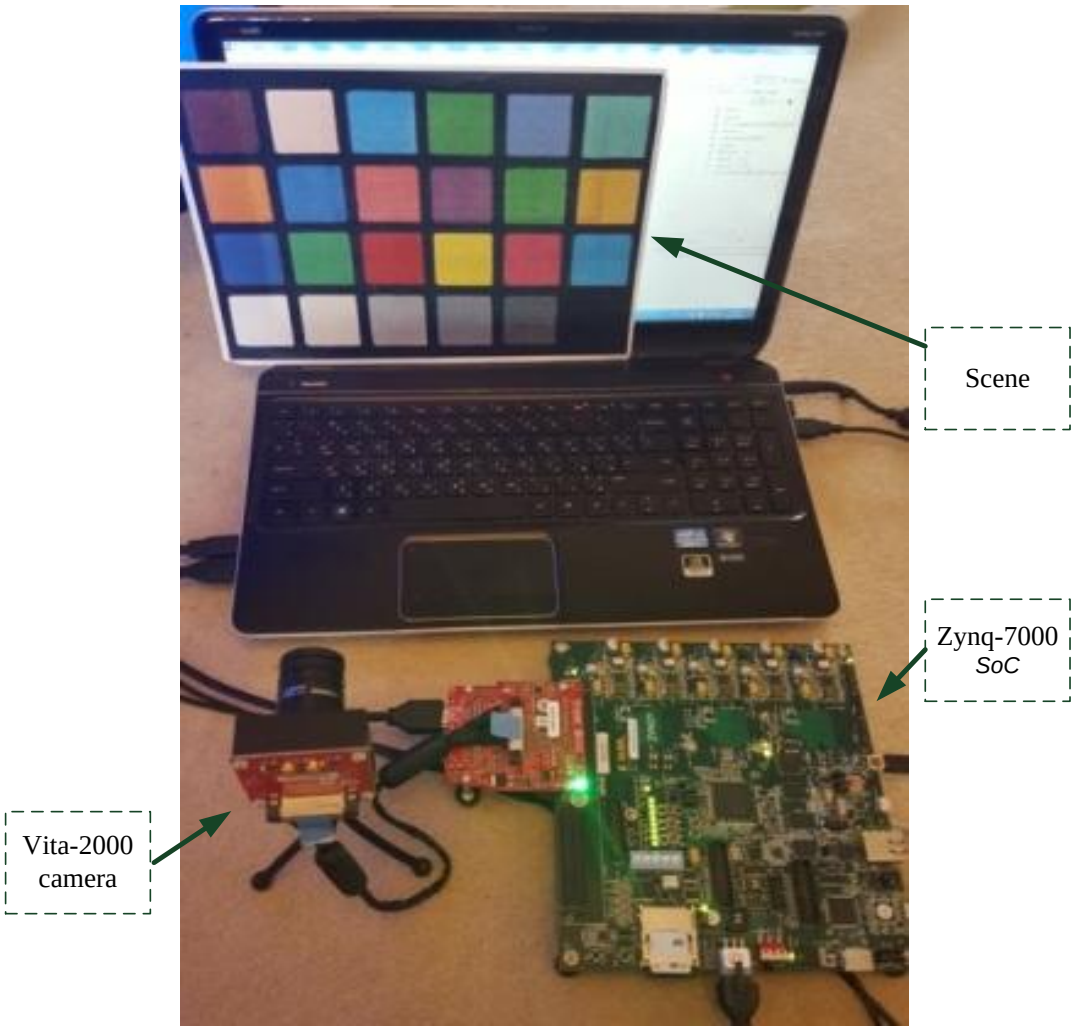


Figure 3.10 Testing platform components

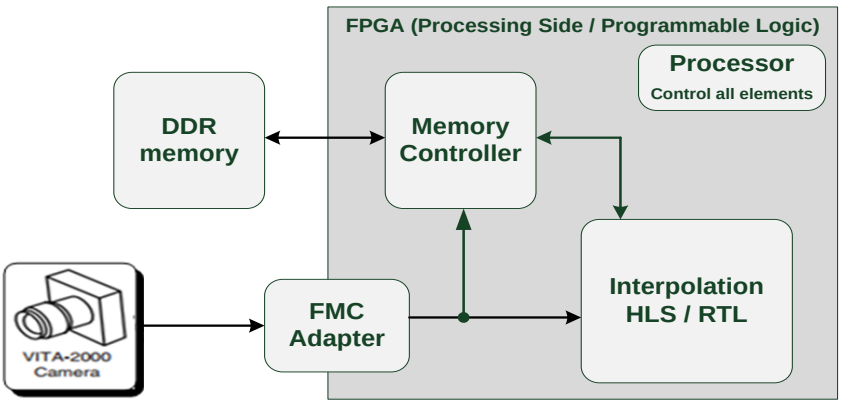


Figure 3.11 HLS/RTL testing platform

3.3.1 Performance and Resources Utilisation

In the RTL-based implementation, the maximum operational frequency is 209.6 MHz, but the system was tested at a frequency of 200 MHz due to the limitation in memory interface. When the maximum frequency is used, the throughput reaches 419.2 MP/s. The latency of the design is 7720 clock cycles (four lines + 80 pixels), as the data passes through three different sets of FIFOs to process the data. The first set of FIFOs for G interpolation requires approximately 3840 clock cycles; the RB interpolation FIFO set needs 1920 clock cycles; and finally the median filter FIFO requires approximately 1920 clock cycles. The other logic needs approximately 40 clock cycles. The total required time to process the entire image of size 1920x1080 is equal to 8.53 ms, including the memory interface timing, which is the bottleneck in this design. This implementation can process 117 frames per second. In contrast, the maximum achieved frequency of HLS-based implementation is 176 MHz. Due to the use of the memory controller provided in the Zynq SoC, the maximum frequency has been used in testing the design. The HLS-based implementation achieved a throughput of 352 MP/s. The implementation latency is similar to the RTL implementation. The time measured for processing an image of size 1920x1080 is equal to 5.89 ms. This difference between the two implementations corresponds to the difference in the memory interface. The implementations can perform better than the designs proposed in [99, 101, 118, 131] in terms of execution time and average throughput, as shown in Table. 3.1 for images of size 1920x1080. The work in [101]

highly depends on block memory to process the algorithm. Because of that, it cannot process large image sizes with small FPGAs.

Table 3.1 HLS/RTL Implementation Performance against Other FPGA-Based Implementations

Implementation	Execution Time (ms)	Throughput (Mpixel/s)
Multi-core DR Freeman implementation [118]	8.92	232.2
FPGA Bilinear Implementation [99] - (without memory interface)	13.82	150
Bilinear Inter. Architecture for Vision Systems [101] (no specific image size)	4.2 (1024*1024)	250
Xilinx CFA v.7.0 (1ppc) [131]	11.52 (depends on frequency- 1 ppc)	124.4
RTL-based (2 ppc) Adams-Hamilton Implementation	8.53	419.2 (no memory interface) 243.1
HLS-based (2 ppc) Adams-Hamilton Implementation	5.89	352

In terms of resource utilisation, Table 3.2 shows the system resource utilisation for both implementations and for other implementations. It shows that our implementation has a reasonable and better resources utilisation compared to others. Moreover, based on the experimental results, RTL-based implementation has a more optimised design compared to the HLS-based design. This is expected; since designing the application from the gate level provides a greater likelihood to have a more optimised design rather than from higher levels. In RTL-based implementation, BRAMs are mainly used in "Data in buffers" and "Data out" components to buffer the extra data not yet sent to the next stage, due to the difference in data size between the two consecutive components. To solve this problem, a high priority is given to the writing operation over the reading from memory to make sure that the buffers are not full. In HLS-based Zynq implementation, No buffers are used due to the existence of the memory controller in the processor side. For simplicity, Video Direct Memory Access (VDMA) is used to move the data from memory to the design logic. Moreover, the use of ARM processor leads to a reduction in the total resources used in the PL side because no soft processor is needed in this case.

Table 3.2 Resource Utilisation of RTL.HLS Implementations - 2 Pixels/Clock Cycle

	Resources	Data in Buffer	Interpolation engine	Data out buffer	Processor and Mem interface	Total	Percentage
RTL (Virtex-6)	**FF**	90	3125	236	9933	13384	4.44%
	LUT	228	4195	294	7406	12123	8.04%
	Slices	101	1003	202	3163	4469	11%
	BRAM-36	15	11	36	29	88	21%
	Max Freq.	209.6 MHz					
RTL (zynq)	**FF**	-	3110	-	PS - ARM processor	3110	2.90%
	LUT	-	4160	-	PS - ARM processor	4160	7.81%
	BRAM	-	11	-	PS- ARM processor	11	7.85%
	Max Freq.	206.7 MHz					
HLS (zynq)	**FF**	-	5863	-	PS- ARM processor	5863	5.51 %
	LUT	-	5336	-	PS- ARM processor	5336	10.03 %
	BRAM-36	-	17	-	PS- ARM processor	17	12.14 %
	Max Freq.	176 MHz					
Xilinx CFA v.7.0 without filter (1ppc) [131]	**FF**	-	3414	-	-	-	-
	LUT	-	3112	-	-	-	-
	BRAM-36	-	5	-	-	-	-
	Max Freq	180 MHz ----- With extra 16 DSP logic					
FPGA Bilinear Implementation RTL (1ppc) [99]	**FF**	-	72	-	-	-	-
	LUT	-	658	-	-	-	-
	BRAM-36	-	7	-	-	-	-
	Max Freq	171 MHz					
Bilinear Inter. Architecture for Vision Systems [101]	**Slices**	-	362	-	-	-	-
	DSP48	-	3	-	-	-	-
	BRAM-18	-	256 (High)	-	-	-	-
	Max Freq	133- 250 two versions					

3.3.2 Power Consumption

This section investigates the power consumption of the implemented designs. No work in the literature discusses the power consumption of such designs with similar functionality, but few works have some similarity in terms of the architecture or the design methods. In [132], the power consumption of a proposed adaptive median filter has been measured. The architecture in this work uses a sliding window for computing the median values, similar to our work. The Xilinx Xpower tool has been used to estimate the power consumption of the standalone median filter hardware. The implementation consumes between 30 and 40 mW for the median filter part. This value is higher than the value obtained by our design, which has a functionality that is more complex. In [133], the edge detection algorithm has been implemented using Vivado HLS, and the corresponding power consumption has been measured on various FPGAs. The author compared the power consumption of the implemented design with the work proposed in [134]. The implementation of HLS-based edge detection consumes 402 mW. This design consists of the edge detection part as well as other parts that form the testing platform. This value does not represent the power consumption of that specific module; rather, it signifies the complete testing platform. The design showed the change in the total consumed power if higher frequency is used.

The following methodology is used to evaluate the power dissipation of HLS-based and RTL-based implementations. In our work, the power consumption has been measured using two different methods to verify the overall results. In the first method, the power information is read directly while the system is working. In this method, the system monitor and ChipScope Pro tools are used to access the information via the JTAG in Vertix-6 [135]. In Zynq-7000 AP SoC, the TI Fusion Power Designer tool is used to monitor the power information on the Zynq board [136] (See Figure 3.12). This tool communicates with the embedded power regulators and PMBus controller inside the Zynq board over TI serial bus to fetch the power information or control it (See Figure 3.13). The second method extracts the power information from the output files of the processed design using power analysis

tools such as the Xilinx Xpower tool [137] for Virtex-6 and the power analysis tool part of the Vivado suite for 7 series FPGAs[138]. After obtaining measurements from the two methods, a comparison is made to evaluate the power dissipation of the RTL and HLS-based implementations.

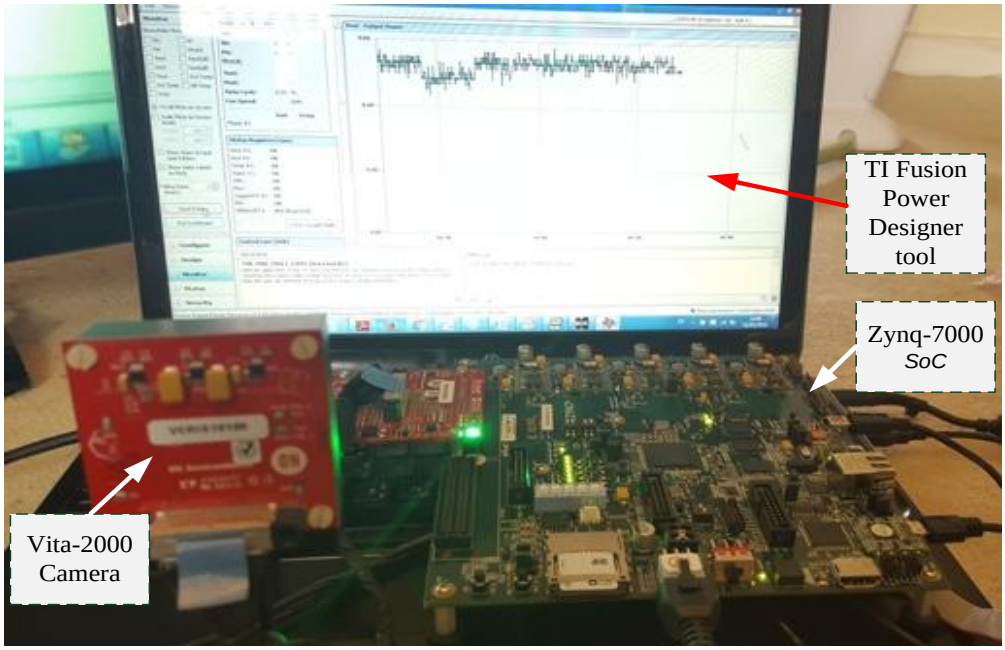


Figure 3.12 TI Fusion Power Designer tool

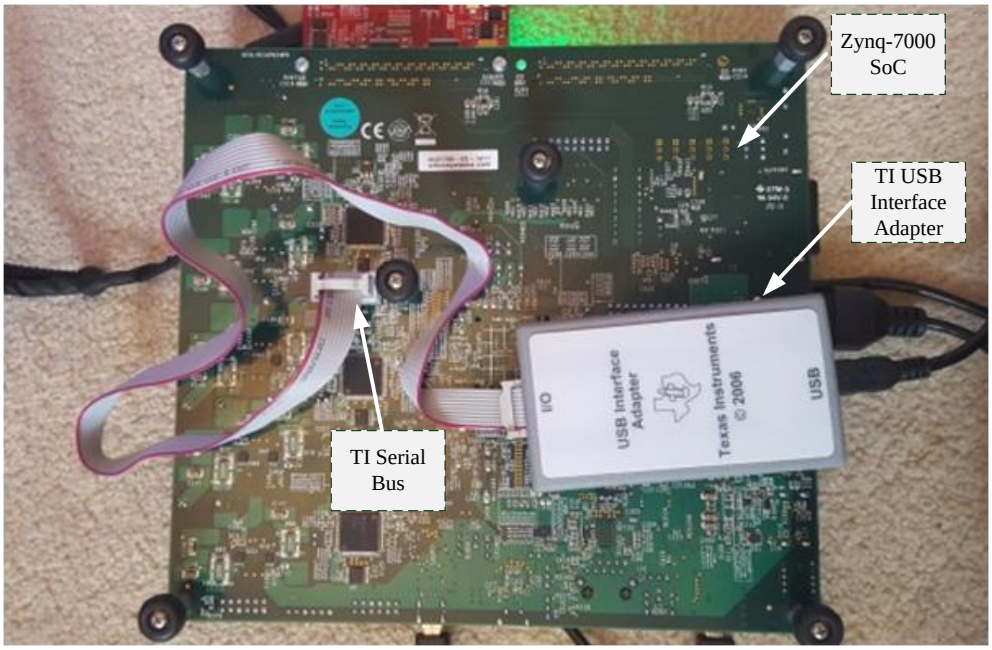


Figure 3.13 TI USB Cable Connection

In Zynq-based implementations (RTL or HLS), the design is split into two parts between the PS and PL sides. The PS side consumes more power compared to PL in this implementation for two reasons. First, the clock resources are generated in the PS side, and second, the DDR memory is attached directly to the PS side. These two resources consume a significant amount of power in any design. The PL side has only the algorithm logics such as FFs, LUTs, BRAMs, and DSPs. Figure 3.14 shows the power consumption of the internal logic circuit of the PL side of the Zynq board for the proposed HLS-based implementation using the Fusion power designer tool. The middle part of the figure shows the power consumption when the design is processing data (dynamic), while the rest shows the consumption at idle state (static). Due to the low polling rate in the tool, the reading cannot be obtained in the normal case. The algorithm has been repeated 1000 times to obtain these readings. For the PS side, Figure 3.15 shows the power consumption of the PS internal logic circuit including AXI buses, I/O, and the processor logic. The power dissipation is approximately 340 mW, with a possible +60 mW based on the processor activities. The DDR and clock generator consume 889 mW and 350mW, respectively. The total power is 1.818 mW for PS and PL, including the static power.

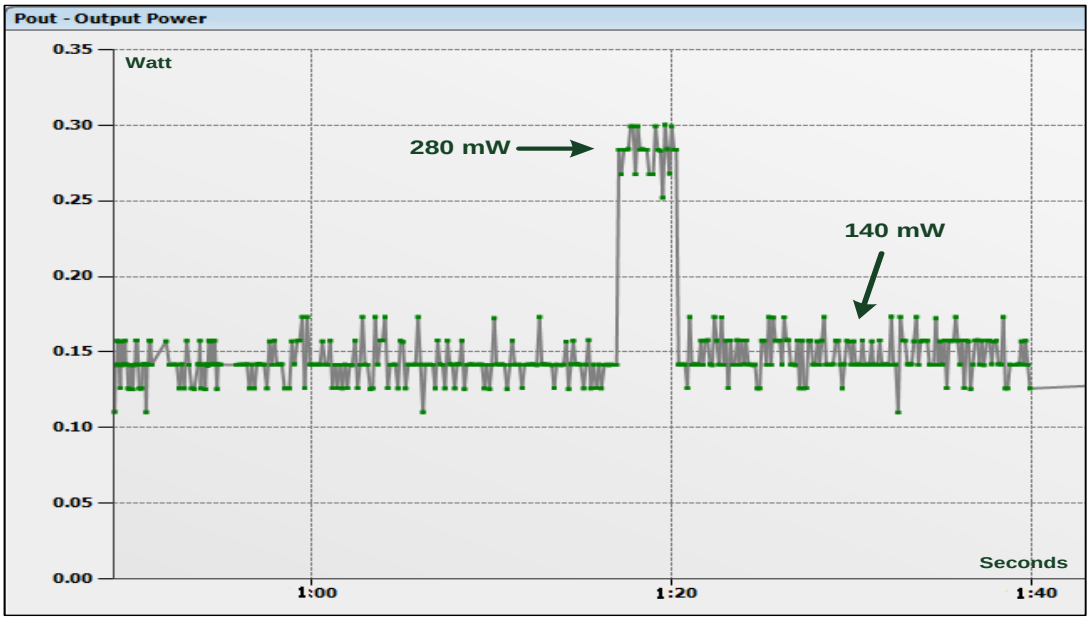


Figure 3.14 Power consumption of the PL side for the proposed HLS-based implementation in the Zynq board (1 ppc)

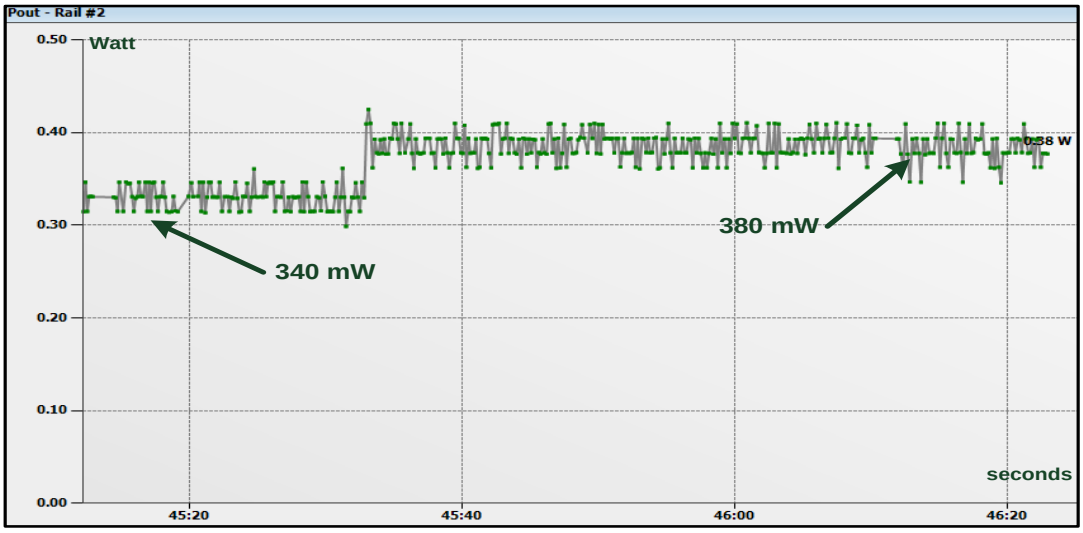


Figure 3.15 Power consumption of the PS side for the proposed HLS-based implementation in the Zynq board (1 ppc)

Similar results have been obtained from the Xilinx power analysis tool. Unlike the previous tool, this tool has no information about the power in each stage of the algorithm or the idle state. Table 3.3 shows the power consumption of the HLS and RTL-based implementations in Virtex-6 and Zynq board using Xilinx power analysis and Xpower estimator tools. The results here are better than the results obtained in the discussed similar works.

Table 3.3 PL- Power Consumption of RTL and HLS Implementations Based on Zynq SoC and Virtex-6 (1 ppc)

Implementation	Stand-alone algorithm logic (mW)	Processor (mW)		Memory Interface (mW)	Logic Total Power (mW) (including Static)
		With clock generator	Without clock generator		
RTL-based Virtex-6	W/O buffers (75) With buffers (150)	Soft processor 219	Soft processor 5	Soft interface 170	539 (3626)
RTL-based Zynq-7000	26	690	340	889	75 (228)
HLS-based Zynq-7000	31	690	340	889	89 (250)

Based on the above table, RTL-based implementation on the Zynq board shows less power consumption compared to the HLS-based one on the same board. This is due to the ability of implementing a more-optimised design when working in the gate level rather than a higher level of abstraction. Figure 3.16 shows the power consumption difference between the two implementations in the Zynq board, including the static power. In Virtex-6 implementation, the design consumes more power compared to the Zynq-7000 implementation. This is because the Virtex-6 has no specialised processor elements or memory interface. Therefore, these elements should be implemented on the logic of the board, which increases the power consumption for the total design.

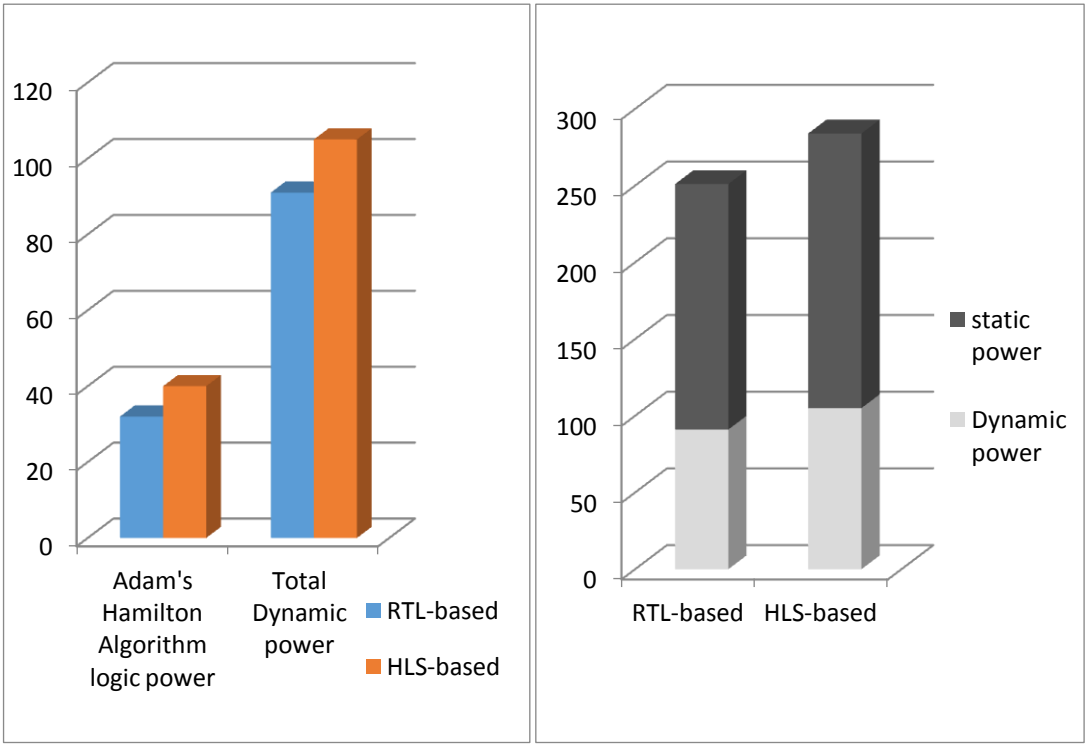


Figure 3.16 PL-Power Consumption of RTL and HLS-based in Zynq-7000 AP SoC (2ppc)

3.3.3 Image Analysis

Figure 3.17b shows the constructed images using the implemented system. The original raw format images are shown in Figure 3.17a. To measure the quality of the

constructed images, a set of images [139] is modified and mosaiced by removing two colours from each pixel to construct images similar to the one produced by the digital camera's sensors. The position of the colours determines the type of Bayer pattern. RGGB Bayer patterned images are produced and then passed to the implemented system stages. The reconstructed images are compared with the original RGB images after measuring the PSNR of the two images. PSNR is used to the measure how closely the constructed image fits the original one. In our case, a set of three images is mosaiced using a MATLAB code. The three Bayer patterned images are passed to the implemented system, and PSNR values are calculated for the constructed images to show the algorithm quality. Table 3.4 shows the PSNR values for the output images as well as the PSNR values for the same images reconstructed using the function demosaic from MATLAB's image processing toolboxes. The results show that our system can give similar quality to the gradient-corrected linear interpolation algorithm, which is used in the default demosaic function in MATLAB. This result matches the PSNR values extracted for the same algorithm in the evaluation studies conducted in [121-123].

Table 3.4 Computed PSNR Values for the Constructed Images vs. MATLAB

Image	**PSNR of the implemented system images (dB)**	**PSNR of images using demosaic's Matlab function (dB)**
Image1 (woman)	34.2142	34.5622
Image2 (Battle)	41.931	36.3618
Image4 (Green mountains)	31.254	32.4580

(a) **(b)**

Figure 3.17 (a) Raw Bayer images (b) Constructed images using the design

3.4 RTL vs. HLS - Evaluation and Comparison

This section discusses the differences between the two approaches. The comparison is based on the experimental results extracted from this work and other research works in the literature. The main aspects of comparison are throughput, frequency, area optimisation, power consumption, automation process, and development time.

In terms of area optimisation, Table 3.2 summaries the resources utilised by each approach. It shows that the HLS-based implementation utilised more resources than

the RTL-based one. The main reason for this observation is that the designer has a full control over the parts of the RTL-based design. In contrast, the HLS-based approach is an automated process that converts the high-level code of the method descriptions to low-level abstraction. Moreover, the HLS tool controls the synthesis process through optimisation directives that allow the creation of high-performance and area optimised hardware implementation. However, in the normal case, with all these features, this automated process cannot represent the higher descriptions in an optimised way as the RTL designer. Many works in the literature support this outcome and observation, including [140], in which the HLS-based implementation resource usage is 3-4 times higher compared to RTL-based. In contrast, the HLS-tool could produce an implementation with a better optimisation compared to the RTL-based one developed by a novice RTL designer. However, Xilinx document [141] shows that the HLS-based implementation that uses floating-point operations performs better and uses less resources compared to the RTL-based.

In terms of throughput and maximum frequency, Tables 3.1 and 3.2 summarise the outcomes of these two aspects. Throughput and frequency have a strong relationship with area utilisation. The throughout and frequency are determined by the amount of parallelism and the iteration interval, respectively. To achieve high throughput from a design, the amount of parallelism should be increased. This means increasing the resource usage of the FPGA. Alternatively, achieving higher frequency can be done by applying more intermediate registers to split the implementation into small flow sections to decrease iteration intervals, which leads to an increase in the resource utilisation. Based on Tables 3.1 and 3.2 and for the same amount of parallelism, RTL-based implementation achieved higher throughput and higher frequency compared to the HLS-based implementation. Balancing between the area and throughput gives RTL-based implementation the advantage again.

In terms of power consumption, the more resources used, the greater amount of dynamic and static power is needed by the design. Table 3.3 and Figure 3.16 support this outcome. HLS-based implementation has more total power consumption due to a difference in the used resources between the two implementations. The types of the

used resources in such designs could have an impact on the power consumption. In terms of process automation and development time, HLS-based implementation cannot be compared to the RTL-based in any way. Developing the design using C/C++ is much easier than in HDL. Moreover, the user does not need to look deeply in the logic-level of the FPGA, which simplifies the process. In HLS tools, the design can be interfaced easily with many types of interfaces, while the RTL-based implementation needs more effort to do such work. Finally, for development time, the total time includes the implementation time plus the testing and debugging time. For RTL-based implementation, any simple design can take a relatively long time to be developed, so imagine the time needed to develop complex designs. Alternatively, HLS-based implementation needs relatively the same effort of developing any algorithm using high-level languages. For our implementations, the time needed for developing the RTL-based implementation was two months, while the HLS-based implementation was developed in one week. In [141], a radar design that uses floating-point has been implemented using the RTL and HLS approaches. The paper showed that the HLS-based design has been implemented in one week compared to 12 weeks for the RTL approach. Testing and debugging in HLS is faster than in the RTL approach, as revealed by the work carried on this chapter and based on the Xilinx document [141]. Table 3.5 summaries the comparison between the RTL and HLS approaches in terms of the above criteria.

Table 3.5 RTL vs. HLS Approach comparison

Criteria	HLS-based approach	RTL-based approach
Area optimisation	Good	Excellent
Throughput	Good	Excellent
Power consumption	Good	Excellent
Process automation	Excellent	Poor
Development time	Excellent (1 week)	Poor (8 weeks)

3.5 Summary

Modern cameras require high-performance architectures to process the data within a small amount of time due to the rapid increase in input data for quality purposes. In

the past, the size of each image frame was limited to a few megabytes. Currently, each frame could reach 15-20 times the size of the old frames. Moreover, the numbers of processing components are increased due to the increase in image processing techniques.

This chapter presented the design and architecture of an efficient implementation of the Adams-Hamilton interpolation algorithm that constructs the full image by estimating the missing colours in each pixel. The architecture exploited the parallelism feature provided by FPGAs to increase the performance of the algorithm to meet the processing requirements of modern cameras. In addition, the architecture is implemented using two different approaches available on the FPGA market: the RTL-based approach and the HLS-based approach. The experimental results of both implementations show good throughput, lower execution time, and reduced power consumption compared to the current available solutions. A comparison and evaluation study has been carried out between the two approaches to determine the best practice for each approach. The study showed that the RTL-based approach is better in the case when the time and effort are not critical aspects compared to the need of a highly optimised design in terms of area, power, and performance. In contrast, the HLS-based approach is a very good choice for limited time projects. The HLS-based approach can still provide high-performance designs with higher area utilisation.

Chapter 4 : Dynamic Partial Reconfiguration Implementation for Automatic White Balance on FPGA

4.1 Introduction

Colour constancy is the ability of the eye to keep the perceived colour of objects relatively constant under varying illumination conditions. This feature is part of the human colour perception system. In contrast, the devices that capture images, such as digital cameras, cannot keep the colour of objects constant under different illumination conditions. Each pixel value recorded by the sensor is related to the colour temperature of the light source. The image will appear reddish when the object is illuminated with low colour temperature light. In contrast, it will appear bluish if a high colour temperature is used [89]. For the above reason, a white balance algorithm is required to process the image data so it looks constant under different light sources. Two different types of white balance algorithms are discussed in the literature: the first type compensates for predefined parameters in the algorithm based on the known light source temperature. The second type, known as AWB, automatically expects the light source temperature and based on that, it calculates the parameters to be applied in the algorithm. Different algorithms have different image qualities and vary in terms of computation demands, but in general, all algorithms require intensive computation. In section 2.2.1.2, many AWB methods have been discussed in general. Many evaluation and comparative studies, such as [89, 142], have been carried out for different AWB algorithms to find the best-fit method for most applications.

A number of AWB algorithms have been proposed in the literature, such as GWA [90], PRA [91], Retinex theory, standard deviation-weighted gray world, and other algorithms in which the most likely illuminant from a predefined set using correlation is estimated. The correlation-based methods are computationally

intensive methods and are not preferable for real-time applications. In contrast, gray world and perfect reflector/retinex are simple algorithms and result in a linear correction to the R and B channels. With the gray world algorithm, the assumption is made that the average of any image is gray, and the probability of red, green, and blue colours are equal. This algorithm is performed by calculating the mean values of the three channels, which are then used to calculate the ratio between them based on the green channel. Finally, the obtained correction values are used to correct all pixels in the image. This algorithm works fine with the colourful images but not the images that have a dominant colour such as sky or sea, etc. The second theory (retinex/perfect reflector) assumes that the best reflectors can be used as the reference value of the white colour in the image. The white balance is obtained by finding the correction coefficients of the red and blue channels while the green channel remains unchanged. This theory does not work well if the image is colourful, because in that case a shift in the entire colour range may occur. For the above reasons, a new algorithm has been proposed to combine the two theories and their advantages. The algorithm is called gray world and retinex theory [143]. Its idea is to find coefficients that satisfy both algorithms to correct the red and blue channels.

White balance algorithms can be performed in a software environment, but when throughput and latency is a design issue, a customised hardware implementation is preferable. The hardware implementations can be done in FPGAs or ASICs. Although the ASIC is considered as a low cost and low power consumption environment, the FPGAs can provide more flexibility to the users. The DPR feature can provide additional flexibility to the design by changing the functionality of part of the design while the rest of the design is working. This feature can reduce the total required resources and the power consumption for a specific system. Accessing the FPGA configuration memory is performed internally through the ICAP.

Few research studies discuss the implementation of auto white balance algorithms in hardware environment. In [108], a hardware-software co-design has been implemented to reduce the resource utilisation as much as possible while keeping the performance reasonable. In [102], a static design for the gray edge hypothesis is implemented, but no specific information about the performance and the resource

utilisation was provided. Xilinx provides a white balance application part of the Xilinx LogiCORE Image statistic core in [144] and part of [105]. In [145], a static implantation for new proposed algorithm is described. Only the resources utilisation report is presented with no performance measurements. In [146], an Altera-based static architecture for white-balance method part of enhancement stage of image signal processor is presented. Similar implementation is proposed in [147] .The paper showed reasonable results for real-time image processing. None of these research efforts attempts to apply the DPR to such designs to reduce the power consumption and resource utilisation and to increase the performance.

This chapter presents a novel DPR implementation of auto white balance, which minimises the used resources, reduces the power consumption, and increases the performance. The design of the different components of the data path is discussed and analysed. The chapter also presents a brief introduction of the algorithm and compares implementation results to other similar designs.

Table 4.1 Light Source Temperatures

Colour Temperature	Light sources
10000 – 15000 K	Clear Blue Sky
6500 – 8000 K	Cloudy Sky/ Shade
6000 – 7000 K	Noon Sunlight
5500 – 6500 K	Average Daylight
5000 – 5500 K	Electronic Flash
4000 – 5000 K	Fluorescent Light
3000 – 4000 K	Early AM/ Late PM
2500 – 3000 K	Domestic Lighting
1000 – 2000 K	Candle Flame

4.2 Gray World and Retinex AWB Algorithm

A white balance algorithm is required to process the images so it looks similarly under different light sources. Table 4.1 shows different light sources and their temperature ranges. Gray world and retinex—Lam's algorithm—can be considered as one of the usefull algorithms that deal with images in terms of auto white balancing. Based on the evaluation and comparison studies in [89, 142], Lam's AWB is chosen for the targeted application due its overall good performance and low implementation

and computation complexity. Moreover, Lam's algorithm showed good results in the subjective and objective evaluations conducted in these studies between different AWB algorithms. Subjective evaluation methods are based on the visual evaluation and the human perception of the colour in a scene. Objective evaluation methods are based on computed numerical metrics describing the properties of the images, such as colour difference metric, by calculating the difference between a faded image and a reference image.

Lam's algorithm combines the gray world and retinex theories. It finds coefficients that satisfy both theories. Gray world assumption uses coefficients based on the mean value of the red, green, and blue channels, and then the coefficients are used to correct the pixel values. The Green channel stays unchanged because it is used as a base channel. Equations (4.1) and (4.2) show the algorithm when the Red channel is corrected, where $\hat{\alpha}$ is the correction coefficient, $\hat{I}_r$ $\hat{I}_b$ are the new pixel values using the theory and I_r, I_g, and I_b are the red, green and blue pixel values. Equations (4.3) and (4.4) show the Blue channel correction.

$$\hat{\alpha} = G_{avg} / R_{avg} \quad (4.1)$$

$$\hat{I}_r (x, y) = \hat{\alpha} * I_r (x, y) \quad (4.2)$$

$$\hat{\alpha} = G_{avg} / B_{avg} \quad (4.3)$$

$$\hat{I}_b (x, y) = \hat{\alpha} * I_b (x, y) \quad (4.4)$$

In contrast, retinex theory uses coefficients based on the maximum values in each channel to correct the pixel values. Equations (4.5) and (4.6) show the algorithm when the red channel is corrected, where $\hat{\alpha}$ is the correction coefficient, $\hat{I}_r$ $\hat{I}_b$ are the new pixel value using the theory and I_r, I_g, and I_b are the red, green and blue pixel values. Equations (4.7) and (4.8) show the Blue channel correction.

$$\hat{\alpha} = \max \{I_g (x, y)\} / \max \{I_r (x, y)\} \quad (4.5)$$

$$\check{I}_r (x, y) = \hat{\alpha} * I_r (x, y) \quad (4.6)$$

$$\hat{\alpha} = \max \{I_g (x, y)\} / \max \{I_b (x, y)\} \quad (4.7)$$

$$\check{I}_b (x, y) = \hat{\alpha} * I_b (x, y) \quad (4.8)$$

Lam's algorithm combines the above equations to produce new coefficients that satisfy both theories outputs. The use of mean values makes the algorithm suitable for colourful images, and the use of the maximum values allows the algorithm to deal better with images with dominant colours. Combining the equations makes the algorithm deal better with different cases. Equations (4.9) to (4.18) show the new set of equations when the combination is applied. To change the red channel's pixel values, (4.5) is applied. Equation (4.6) is used for the blue channel correction, where (μ_r, γ_r) and (μ_b, γ_b) are the coefficients used for correcting the image pixels in the auto white balance theory and $\check{I}$ is the new pixel value for red and blue pixels.

$$\check{\boldsymbol{I}}_r(x, y) = \mu_r I^2_r(x, y) + \boldsymbol{\gamma}_r I_r(x, y) \quad (4.9)$$

$$\check{\boldsymbol{I}}_b(x, y) = \mu_b I^2_b(x, y) + \boldsymbol{\gamma}_b I_b(x, y) \quad (4.10)$$

To satisfy the gray world assumption in the Red channel, the sum of the new Red channel pixels should be equal to the sum of green channel pixels as calculated in (4.11), and then it is substituted in (4.9) to obtain equation (4.12). For Blue channel, the sum of the new Blue channel pixels should be equal to the sum of green channel pixels as calculated in (4.13), and then it is substituted in (4.10) to obtain equation (4.14).

$$\sum_{x=1}^{M}\sum_{y=1}^{N} \check{I}r(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} Ig(x,y) \quad (4.11)$$

$$\mu\sum_{x=1}^{M}\sum_{y=1}^{N} I^2r(x,y) + \gamma\sum_{x=1}^{M}\sum_{y=1}^{N} Ir(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} Ig(x,y) \quad (4.12)$$

$$\sum_{x=1}^{M}\sum_{y=1}^{N} \check{I}b(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} Ig(x,y) \quad (4.13)$$

$$\mu\sum_{x=1}^{M}\sum_{y=1}^{N} I^2b(x,y) + \gamma\sum_{x=1}^{M}\sum_{y=1}^{N} Ib(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} Ig(x,y) \quad (4.14)$$

To satisfy the retinex theory in the Red channel, the maximum value in the new Red channel should be equal to the maximum value in the green channel as calculated in (4.15), and then it is substituted in (4.9) to obtain (4.16) for the red channel case. For Blue channel, the maximum value in the new Blue channel pixels should be equal to

to the maximum value in the green channel as calculated in (4.17), and then it is substituted in (4.10) to obtain equation (4.18).

$$\max\{\check{I}r\,(x,y)\} = \max\{Ig\,(x,y)\} \tag{4.15}$$

$$\mu\max\{I^2r\,(x,y)\} + \gamma\max\{Ir\,(x,y)\} = \max\{Ig\,(x,y)\} \tag{4.16}$$

$$\max\{\check{I}b\,(x,y)\} = \max\{Ig\,(x,y)\} \tag{4.17}$$

$$\mu\max\{I^2b\,(x,y)\} + \gamma\max\{Ib\,(x,y)\} = \max\{Ig\,(x,y)\} \tag{4.18}$$

The coefficients (μ, γ) are unknown in the previous equations and can be solved analytically using Gaussian elimination [148] or using Cramer's rule [149], as the two equations (4.12) and (4.16) for Red channel and equations (4.14) and (4.18) for Blue channel have two unknowns. The equations can be represented in (4.19) and (4.20) as a matrix form for Red and Blue channels respectively.

$$\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} I^2r & \sum_{x=1}^{M}\sum_{y=1}^{N} Ir \\ \max I^2r & \max Ir \end{bmatrix} \begin{bmatrix} \mu \\ \gamma \end{bmatrix} = \begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} Ig \\ \max Ig \end{bmatrix} \tag{4.19}$$

$$\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} I^2b & \sum_{x=1}^{M}\sum_{y=1}^{N} Ib \\ \max I^2b & \max Ib \end{bmatrix} \begin{bmatrix} \mu \\ \gamma \end{bmatrix} = \begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} Ig \\ \max Ig \end{bmatrix} \tag{4.20}$$

4.3 Hardware Implementation

The algorithm has been implemented using two methods in this chapter: RTL implementation and HLS implementation using the Xilinx HLS tool. VHDL has been used for RTL implementation, while C code has been used in the HLS implementation. The two methods are fully demonstrated to check the difference between them in terms of performance, power consumption, and resource utilisation. The implementations are demonstrated on Virtex-6 for RTL-based implementation and Zynq-7000 AP SoC for both implementations. In the case of Virtex-6, the Microblaze soft processor is used to control the design, while the Cortex ARM

hardwired processor is used in the case of Zynq implementation. The implementations have been compared to other similar designs in the literature.

Based on the DaVinci imaging-processing pipeline from Texas Instruments [72] and other examples, AWB is considered as one of the stages of the typical IPP, as shown in Figure 4.1. The stage can be located after or before the interpolation stage. However, since the CFA image has one third of the RGB image data, the white balance stage is preferred to be processed before the interpolation stage to increase the performance to the double at least and to decrease the resources needed, as two-thirds of the image data did not exist in this stage. In addition to the above reason, the algorithm in this case can produce accurate results due to the use of the original data from the sensors, not estimated data as the case of the interpolated image.

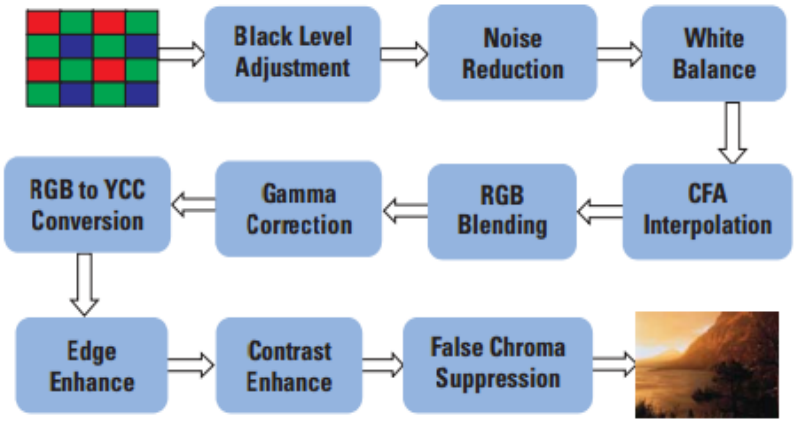


Figure 4.1 Davinci IPP from Texas Instruments [72]

4.3.1 RTL-Based Implementation

The proposed RTL implementation on the Virtex-6 FPGA is illustrated in Figure 4.2 and Figure 4.3. The design exploits the DPR feature to reduce the required resources and consumed power and to increase the performance. The design is divided into two parts: the static part and the reconfigurable part. The static part of the system contains a Microblaze processor, the top-level part of the system, a configuration engine, and the Xilinx NPI-MPMC memory interface [130]. The reconfigurable part of the system contains the aspect of the design that can be replaced regularly to change the system functionality.

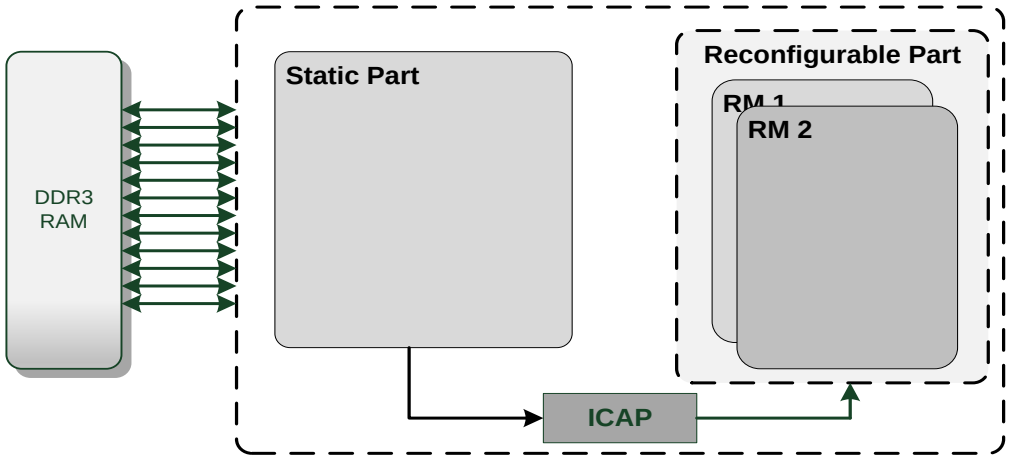


Figure 4.2 Proposed dynamic partial reconfiguration-based system

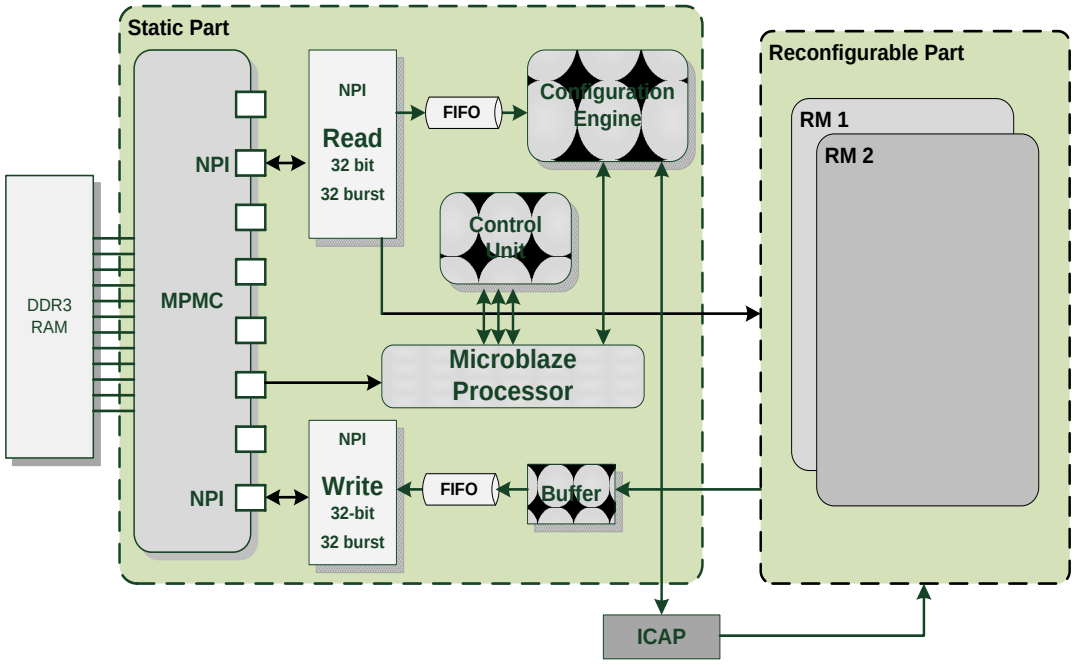


Figure 4.3 RTL-based implementation block diagram

On the other hand, the RTL-based implementation on the Zynq-7000 AP SoC board uses the ARM hard processor and the hardwired memory controller part of the processing system side as an alternative for the Microblaze and MPMC shown in Figure 4.3. This practice decreases the total resource utilisation and the total complexity of the design. The use of DPR requires additional logic to configure the

function modules internally on the fly. The reconfiguration engine proposed in [46] is used to configure the modules.

Static Part

The static part of the RTL-based implementation consists of the following components:

Memory Interface: The Xilinx NPI is used to connect the system to the DDR memory through the Xilinx MPMC. NPI is the fastest and lowest latency connection among other available options through MPMC. The designed NPI-MPMC controller achieved a latency of 30 clock cycles between burst data transfers. The average throughput of using the NPI connection with a frequency of 100 MHz could reach 260 MB/s. This average throughput can limit the overall throughput of the system due to the latency of each transfer from the DDR memory. Two ports of the MPMC are dedicated for NPI, one for reading data and the other for writing. A Microblaze processor is used to control data transfer from/to the memory side and to control the number of required bursts in each transfer.

Control Unit and Design Wrapper: The top design contains the Microblaze processor, which is used to control the flow of data between the DDR memory and the system. Moreover, it controls the configuration engine by triggering the signals that fetch the partial bitstreams from memory and communicate with the ICAP primitive.

Configuration Engine: The RTL-based reconfiguration engine is part of the ICAP controller proposed in [46]. The reconfiguration engine is used to configure the hardware tasks internally on the fly without stopping the other tasks or functions. These tasks or modules are floor-planned according to Xilinx reconfiguration, where each RM can be placed in a single reconfigurable region based on the information in the bitstream associated with each RM. Moreover, at each reconfigurable region, the tool generates an extra 'black-box' bitstream. The black-box removes all the logic in the region apart from the static routes passing. Blanking the reconfigurable region could decrease the power consumed by this part of the design at idle state. The engine exploits the full speed of the ICAP primitive (see Figure 4.4) to configure the

configuration memory with a speed up to 400 MB/s. The tasks are brought from DDR external memory using the NPI through the Xilinx MPMC for the RTL-based on the Virtex-6 FPGA. For the HLS-based engine in Zynq-7000 AP SoC, the embedded memory controller is used for such cases.

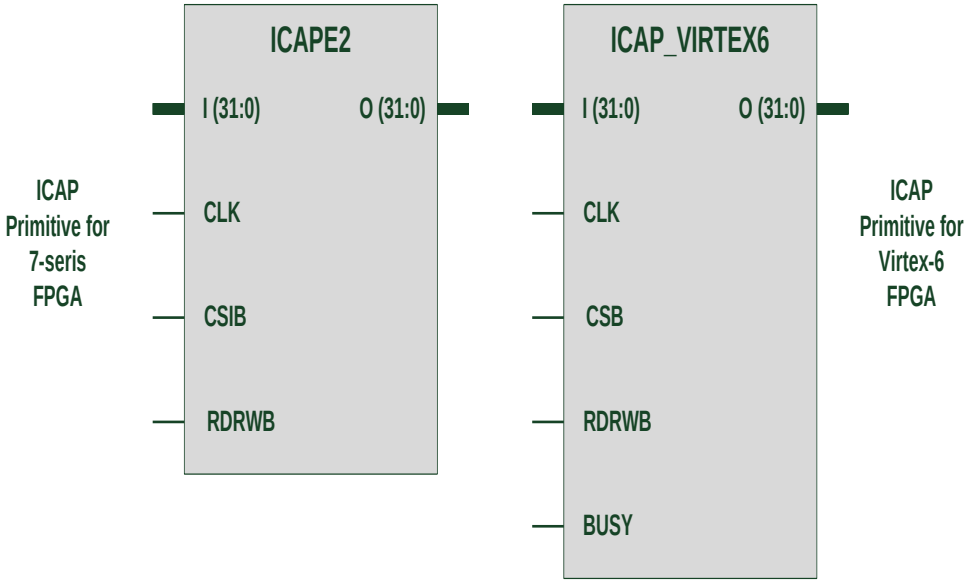


Figure 4.4 ICAP primitive for 7-series and Virtex-6 FPGA

Based on the internal configuration manager proposed in [46], the Virtex-6 configuration engine consists of three components: Finite State Machine (FSM), dual-port BRAM, and the ICAP primitive (see Figure 4.5). The bitstream file does not need any modifications to its content before it passes to the ICAP input port, as all the information and commands for configuration are inside the header of the bitstream file. The FSM internally controls the transfer of data between external memory and the input port with the aid of the read/write buffer in the dual-port BRAM. The data are written in the buffer starting from the first address until the last address of the buffer. Where a jump operation is performed to the first address again in a circular manner until the end of the bitstream file. FSM controls reading of the configuration from the second port and passes the data to the ICAP primitive. The Microblaze should enable the NPI memory interface before the FSM component due to the latency of fetching the data from the external memory.

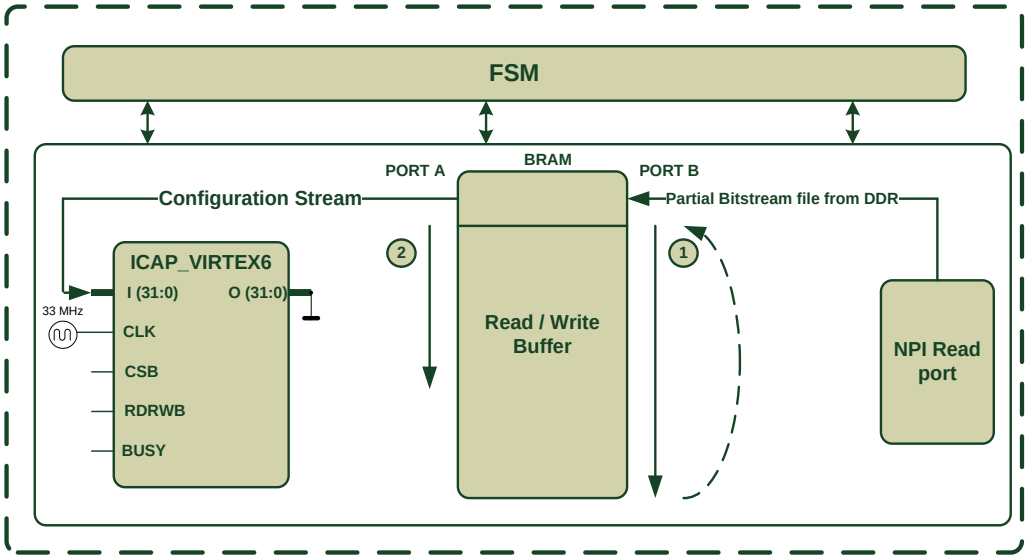


Figure 4.5 RTL-based implementation configuration engine on Virtex-6 FPGA

In Zynq-7000, the configuration engine is implemented in a different way due to the use of the hard memory controller and the ARM processor. The configuration engine consists of two components: the ICAP controller and the DMA. The DMA is controlled by the ARM processor to move the data from the DDR to the ICAP using the embedded memory controller through the high-performance port that connects the PS with the PL side. A maximum burst size is set in the DMA properties to reduce the latency as much as possible. The ICAP controller converts the 32-bit input configuration data to a bit swapped data. Each 8 bits of the data are swapped, as shown in Figure 4.6, and then are passed to the input port of the ICAPE2 primitive. The enable signal is connected to the valid signal from the DMA component to enable the ICAP only when the data are valid (see Figure 4.7).

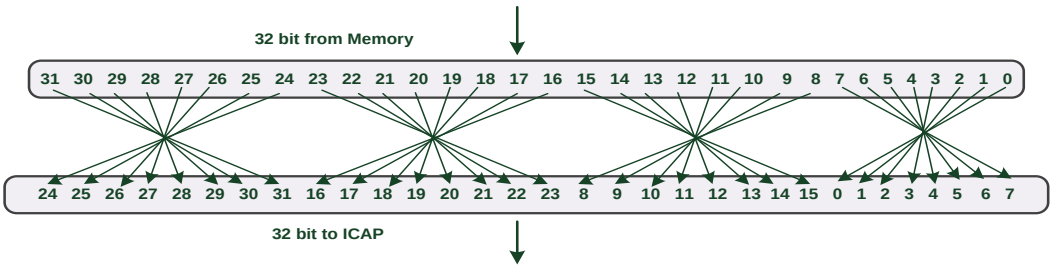


Figure 4.6 ICAP controller bit swapping

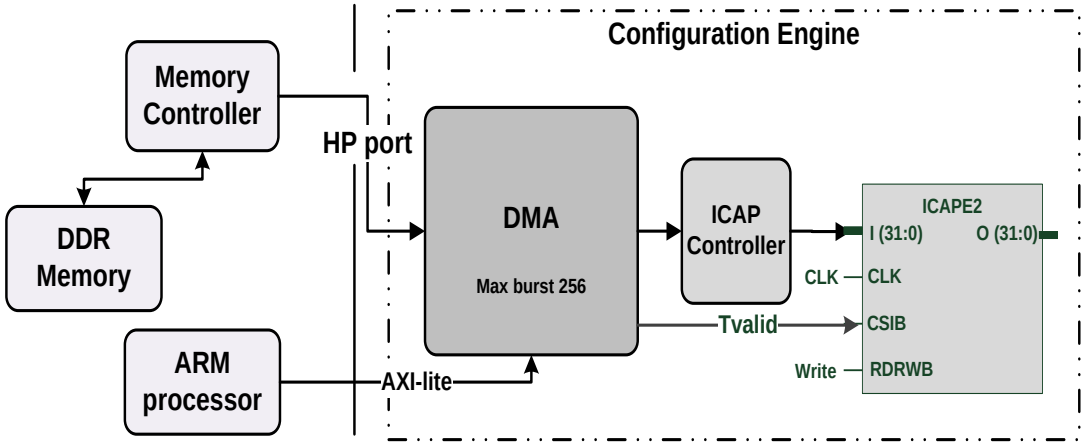


Figure 4.7 RTL-based implementation configuration engine on Zynq-7000 AP SoC

Reconfigurable Part

Before explaining the reconfigurable part, the selected AWB algorithm can be divided into three tasks. These tasks are processed sequentially by scanning the image twice. The first and second tasks are performed in the first scan, while the third task is performed in the second image scan. The first task tries to find the parameters of equations in (4.8) and (4.10). These parameters are specified in equation (4.11):$\sum_{x=1}^{M}\sum_{y=1}^{N} I^2r$, $\sum_{x=1}^{M}\sum_{y=1}^{N} Ir$, $\max I^2r$,$\max Ir$, $\sum_{x=1}^{M}\sum_{y=1}^{N} Ig$, $\max Ig$, $\sum_{x=1}^{M}\sum_{y=1}^{N} I^2b$, $\sum_{x=1}^{M}\sum_{y=1}^{N} Ib$, $\max I^2b$ and $\max Ib$. The second task solves the two equations and finds the unknowns (μ_r, γ_r), (μ_b, γ_b). The third task involves correcting the red and blue channel pixels by substituting (μ_r, γ_r), (μ_b, γ_b) in (4.5) and (4.6).

Based on this division, the system can be implemented using DPR technique by distributing the tasks into two modules. These modules are configured one after the other to reduce the resource utilisation and hence increase the performance. Since the second and the third tasks require floating-point units that necessitate more resources, the system can be divided into two partitions to balance the resource utilisation in each module: the first module consists of the first and the second tasks of the division above, while the second module consists of the third task.

Figure 4.8 shows the hardware implementation of the first task of the first module. In the case of green pixels, the summation of multiplied pixels value $\sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{g}$ and the max $\mathrm{I}^2\mathrm{g}$ value are not required. Solving the equations in the second task can be done using one of two methods: Gaussian elimination or Cramer's rule. The Gaussian elimination method uses the division operation in the first step; therefore, floating-point units are needed in the following steps. Conversely, the division step in Cramer's rule occurs in the last step, which means fewer floating-point units are required in the design. Therefore, a decision was made to choose the Cramer's rule as a method in our implementation. Equations (4.21) and (4.22) show the solution of equation (4.19) using Cramer's rule for the Red channel and equations (4.23) and (4.24) show the solution of equation (4.20) using the same rule. Figure 4.9 shows the hardware model of the rule.

$$\mu r = \frac{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} Ig & \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{Ir} \\ \max \mathrm{Ig} & \max \mathrm{Ir} \end{bmatrix}}{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{r} & \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{Ir} \\ \max \mathrm{I}^2\mathrm{r} & \max \mathrm{Ir} \end{bmatrix}} \qquad (4.21)$$

$$\gamma r = \frac{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{r} & \sum_{x=1}^{M}\sum_{y=1}^{N} Ig \\ \max \mathrm{I}^2\mathrm{r} & \max \mathrm{Ig} \end{bmatrix}}{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{r} & \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{Ir} \\ \max \mathrm{I}^2\mathrm{r} & \max \mathrm{Ir} \end{bmatrix}} \qquad (4.22)$$

$$\mu b = \frac{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} Ig & \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{Ib} \\ \max \mathrm{Ig} & \max \mathrm{Ir} \end{bmatrix}}{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{b} & \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{Ib} \\ \max \mathrm{I}^2\mathrm{b} & \max \mathrm{Ib} \end{bmatrix}} \qquad (4.23)$$

$$\gamma b = \frac{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{b} & \sum_{x=1}^{M}\sum_{y=1}^{N} Ig \\ \max \mathrm{I}^2\mathrm{b} & \max \mathrm{Ig} \end{bmatrix}}{\begin{bmatrix} \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{I}^2\mathrm{b} & \sum_{x=1}^{M}\sum_{y=1}^{N} \mathrm{Ib} \\ \max \mathrm{I}^2\mathrm{b} & \max \mathrm{Ib} \end{bmatrix}} \qquad (4.24)$$

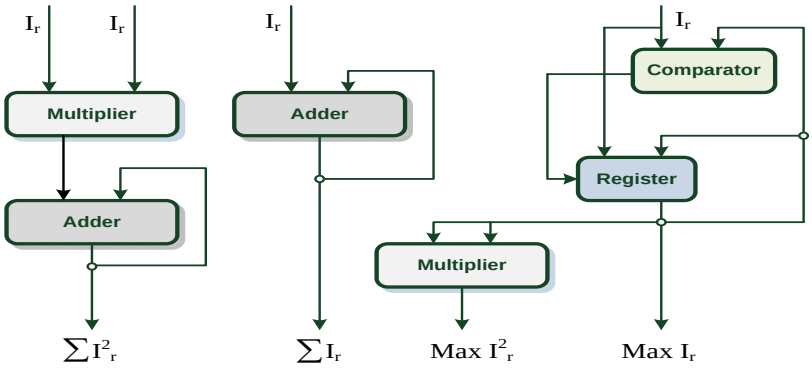


Figure 4.8 Hardware model for the red channel parameters in the first task of the first module

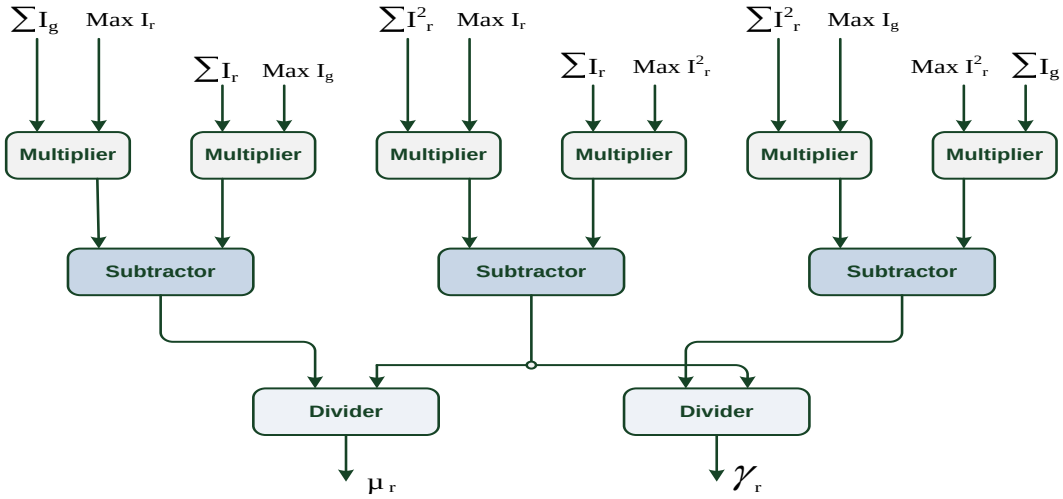


Figure 4.9 Hardware implementation of Cramer's rule

Since the received data from the sensors forms a CFA image, the data is divided into two categories: odd rows and even rows. Odd rows have only the Red and Green pixels, while the blue combined with green pixels are located in the even rows. Therefore, only two channels are processed in the same clock cycle, which decreases the total required resources. The dividers in Figure 4.9 are floating-point units with a single precision, while the other operation units are not. In the third task, floating-point units are used to multiply the values (not integers) of (μ_r, γ_r), (μ_b, γ_b) with the received data to obtain the new pixel's values, as shown in equations (4.5) and (4.6). The Xilinx Core generator is used to generate the floating-point cores in the design. The floating-point operations are spread over the two modules. Theses cores simplify

the implementation by customising the core based on the needed functionality. Figure 4.10 shows how the floating-point cores are connected to each other based on the mentioned equations for the red-colour channel.

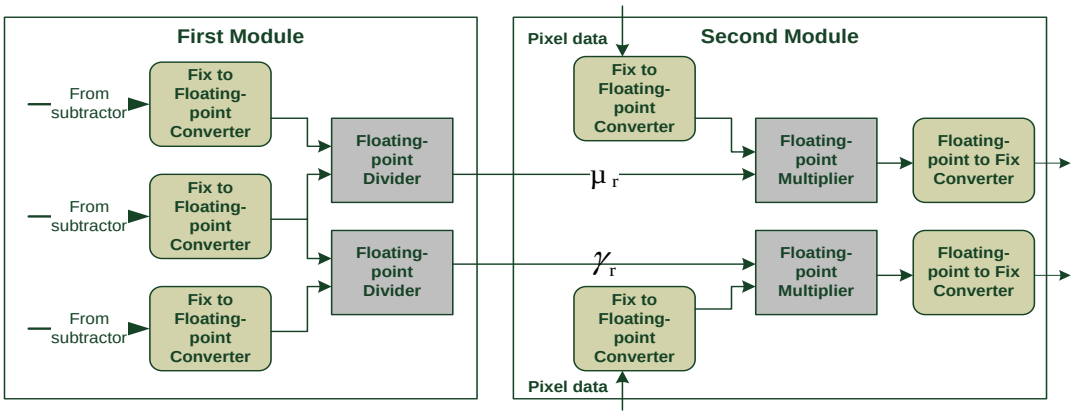


Figure 4.10 Floating-point core deployment in the implementation

Each module is implemented in a different RTL file with the same input and output ports. The files are then integrated into the full design, one at a time, to generate the partial modules. These steps should follow the Xilinx ISE and Vivado DPR flow explained in section 2.1.2.1 to generate the partial bitstreams for the partial modules and for the full design.

4.3.2 HLS-Based Implementation

The second implementation of the algorithm is based on the Vivado HLS tool. The HLS-based design is implemented on the Zynq-7000 AP SoC using the IP-based method. Each module is separately coded using C language instead of describing the algorithm from the RTL-level perspective using VHDL. The C codes are then synthesised by the tool to produce equivalent RTL-level IPs that can be used directly on the design. Similar to HLS-based implementation in section 3.2.2, the algorithm is divided into functions that build up the hierarchy of the RTL-level design. Moreover, the wrapper defines the top-level I/O ports and the used communication protocols. A number of optimisation commands are applied to optimise the design to meet the high-performance requirements. The HLS-based implementation is able to process one or two pixels per clock cycle based on the input data stream width and the width

of data in each function. Unlike the design in section 3.2.2, this implementation does not require any special memory structures, as all equations are based on a single pixel. The hierarchy of algorithm implementation is illustrated in Figure 4.11.

For partial reconfiguration, the configuration engine mentioned earlier in Figure 4.7 is used to configure the modules partially. The engine has been converted to an IP-based engine to be integrated into the system. To apply DPR on the implemented modules, the wrappers of all modules should have the same input and output ports to unify the interface among the RMs in a single reconfigurable region. In Figure 4.11, the modules have two layers: the wrapper layer and the top function layer. In the wrapper layer, all modules should have the same input and output ports. In our case, the ports are input AXI-Stream, output AXI-Stream, and Control Bus AXI-lite. Even if the port is not used in that particular module, it should be included as the case of the output AXI-Stream port in the first module. The control bus in the design is connected to the processor side of the design to control the parameters such as number of columns and rows. Moreover, it is used to send and receive the module data apart from the streaming data.

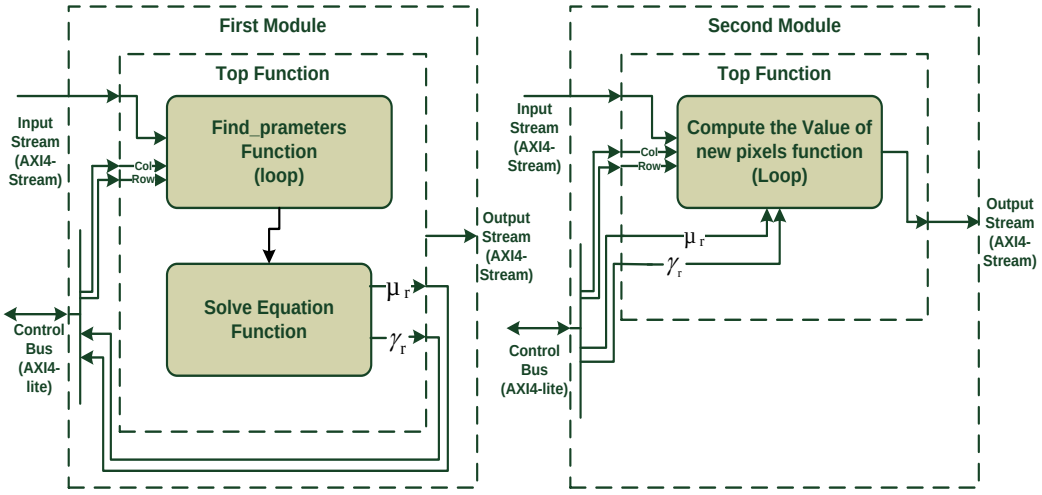


Figure 4.11 Block diagram of HLS-based implementation

The tool converts the C code into an RTL-level circuit. This includes any particular type of components such as BRAMs or DSPs. Due to the existence of division and

multiplication operations that need floating-point components, the tool can generate them automatically based on the equations in an optimised way using special optimisation directives in the tool. On the other hand, in RTL-based implementation, these cores should be generated manually and then should be connected to the other components through the VHDL code.

Similar to the implementation in section 3.2.2, dataflow pipelining is used in the design to enhance the overall performance. This feature allows the sequential functions, loop, or dataflow to operate concurrently at the RTL. The HLS tool adds channels or registers between the operations to hold the intermediate data that flow between the functions to increase the frequency of the system. This feature will increase the total required resources for the implementation. The following command is used to apply this feature in each function and in the main function.

```
#pragma AP PIPELINE II = 1
```

4.4 Experimental Results

The above implementations were executed on a number of FPGA board families with different chips to evaluate their performance. The RTL-based implementation was applied on Xilinx ML605 boards with a Virtex-6 XCE6VLX240T FPGA chip, Virtex-5, Virtex-4, and Zynq-7000 AP SoC. On the other hand, the HLS-based implementation was implemented only on Zynq 7000 AP SoC. Both implementations are able to process up to eight pixels per clock cycle due to the use of the CFA pattern that represents one colour in each pixel. For testing, the data comes directly from camera sensors. The data are either kept in DDR memory for later access or processed immediately by the AWB algorithm (see Figure 4.12). The Vita-2000 camera is used in the testing platform. The camera is connected to the system using an Avent FMC adapter through an FMC connecter. The output images of the system cannot be recognised due to the use of CFA pattern images. For testing purposes, the images are constructed using the interpolation algorithm proposed in the previous chapter to make it possible to recognise the changes in the output image visually after applying AWB.

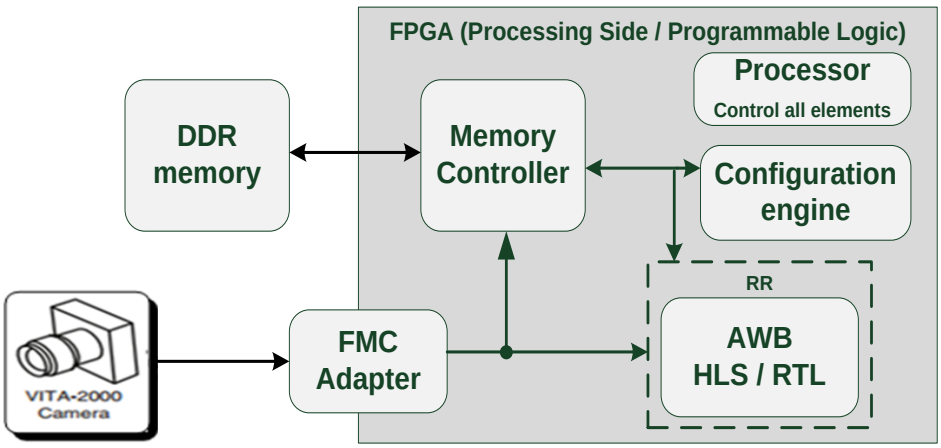


Figure 4.12 AWB -HLS/RTL testing platform

The reconfigurable part of the design has implemented using two different ways based on the number of pixels to be processed in each clock cycle. This method of implementation makes it possible to target either low resource utilisation or high throughput rates. On the other hand, a static design that has no module partitioning has been implemented to compare its outcomes to our DPR implementation.

Targeting low resource utilisation: This method of implementation will reduce the total used resources in the auto white balance implementation compared to the static design. In this design, two bytes -pixels- are processed at the same time, exactly as the static design.

Targeting high throughput rates: This design will increase the throughput of the implementation compared to the static design and the design mentioned above. The design is processing four bytes -pixels- at each clock cycle. This means that the throughput is doubled while the number of used resources is kept in the range of the static design.

4.4.1 Resources Utilisation Analysis

Table 4.2 shows the total utilised resources by the implemented cores and the testing platforms based on Virtex-6 for RTL-based implementation. One of the usages of DPR is to reduce resource utilisation by dynamically time-multiplexing FPGA resources to execute different tasks. In this implementation, because tasks are

executed sequentially over two scans, the existence of some tasks on the FPGA at all times is useless and inefficiently utilises the FPGA resources. DPR overcomes this disadvantage in the static design by having only the functioning tasks on the FPGA. Table 4.2 shows how the DPR can utilise FPGA resources over time by splitting the design into modules. Moreover, time-multiplexing the resources of the FPGA means more resources are available for each task, which enables the designer to increase the parallelism of the design by utilising more resources as shown in the table. Increasing the parallelism leads to a reduction in the execution time for each task and an increase in the overall performance of the system as going to be discussed later in this chapter.

Table 4.2 Resource Utilisation of RTL-Based Cores and Testing Platforms

	Resources	AWB Module 1 + 20%	AWB Module 2 + 20%	Static part	Processor and Mem interface	Total on Chip + 20% of RM
RTL (Virtex-6) Static Design	FF	-	-	19099	9459	19099
	LUT	-	-	18472	7012	18472
	Slices	-	-	6755	3015	6755
	BRAM-36	-	-	29	29	29
	DSP48E1	-	-	34	3	37
RTL (Virtex-6) DPR- low resources utilisation (2ppc)	FF	5018	4148	9933	9459	15955
	LUT	5924	5142	7406	7012	14515
	Slices	1891	1701	3163	3015	5432
	BRAM-36	0	0	29	29	29
	DSP48E1	12	21	4	3	25
RTL (Virtex-6) DPR- High througput rates (4ppc)	FF	9623	9011	9972	9473	21519
	LUT	10263	9395	7450	7032	19765
	Slices	3150	2922	3163	3015	6943
	BRAM-36	0	0	29	29	29
	DSP48E1	20	36	4	3	40
Max Freq.	-	190	203	161.6 MHz		

Table 4.2 shows that the high throughput rate DPR design has approximately similar resource utilisation compared to the static design. These readings support the idea that DPR can conserve many of the resources, compared to the results obtained from the static design as shown in the table, or it can enhance the performance of the design while utilising resources in the range of the static design, as shown in two cases, static RTL and DPR high throughput. The RTL-based on Zynq has the same readings, but with no soft processor and memory interface, as they are part of the embedded processing system side of the chip. Table 4.3 shows similar readings and similar observations but for the HLS-based design on the Zynq-7000 AP SoC. Based on our experiments, the Reconfigurable Region (RR) should have approximately 20% more resources compared to the largest RM for routing purposes.

Table 4.3 Resource Utilisation of HLS-Based Cores and Testing Platforms on Zynq-7000

	Resources	**AWB Module 1**	**AWB Module 2**	**Static Part**	**Processor and Mem Interface**	**Total on Chip + 20% of RM**
HLS (Zynq-700) Static Design (2ppc)	**FF**	-	-	16896	PS side	16896
	LUT	-	-	15684	PS side	15684
	BRAM-36	-	-	14	PS side	14
	DSP48E1	-	-	35	PS side	35
HLS (Zynq-700) DPR- low resources utilisation (2ppc)	**FF**	5618	6669	6061	PS side	14063
	LUT	6124	6965	3863	PS side	12221
	BRAM-36	0	0	14	PS side	14
	DSP48E1	13	22	0	PS side	22
HLS (Zynq-700) DPR- High througput rates (4ppc)	**FF**	8252	9527	6547	PS side	17979
	LUT	9897	9954	4251	PS side	16195
	BRAM-36	0	0	14	PS side	14
	DSP48E1	21	36	0	PS side	36
Max Freq.	**-**	184	199	165 MHz		

Tables 4.4 and 4.5 show the resource utilisation of the static and DPR-based implementations based on number of FPGA families. The number of utilised columns for each component is used rather than the exact number of the utilised components because the system can be used as part of a complete system. The second part of Table 4.5 shows how many resources are saved using the DPR design compared to the static one (e.g. in Virtex-6, the number of CLB columns in the static design is 85 for Virtex-6, while only 68 columns is needed in the DPR design). As shown in the table, employing DPR saves an average of 24.3% of CLBs and around 29% of the DSP components. Figure 4.13 shows the floor planning of the DPR implementation.

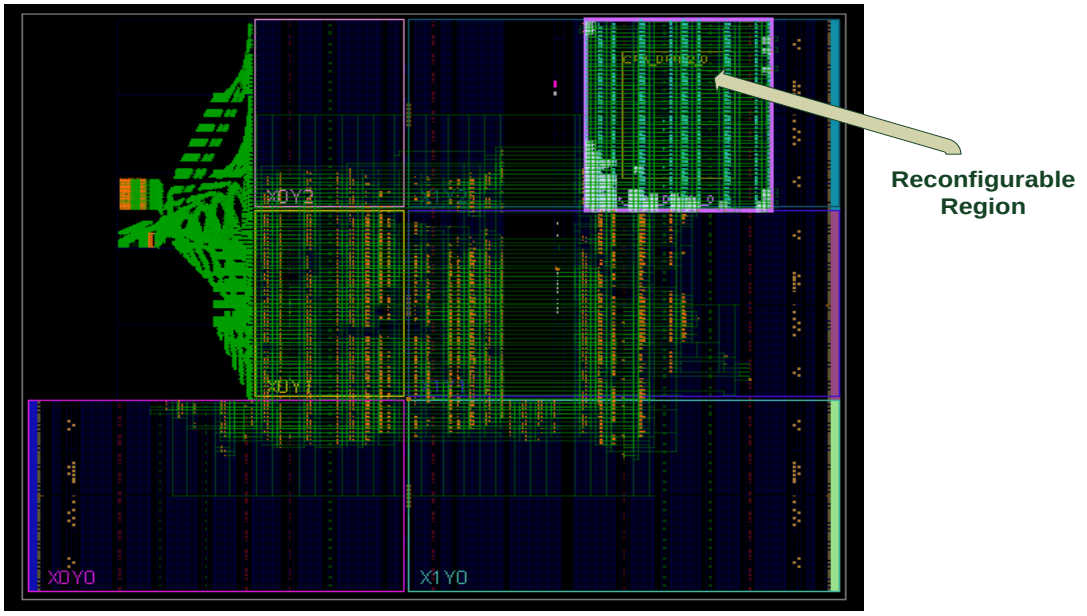


Figure 4.13 DPR implementation floor planning on Zynq-7000 AP SoC

Table 4.4 Column- Based Resource Utilisation of DPR Implementation

Chip	*Static part*		*Reconfigurable part*	
	CLB columns	*DSP columns*	*CLB columns*	*DSP columns*
Virtex-4	95	1	80	6
Virtex-5	88	1	64	6
Virtex-6 RTL	40	1	28	3
Spartan-6	97	1	66	12
Zynq - RTL	14	0	22	3
Zynq - HLS	14	0	23	3

Table 4.5 Resources Saving in DPR vs. Static Implementation

Chip	*Static Design*		*Saving in DPR*	
	CLB columns	*DSP columns*	*CLB Saving*	*DSP Saving*
Virtex-4	205	10	14.6 %	30 %
Virtex-5	197	10	22 %	30 %
Virtex-6 RTL	85	5	20 %	20 %
Spartan-6	202	16	19.3 %	18.75 %
Zynq - RTL	56	5	35 %	40 %
Zynq - HLS	57	5	35 %	40 %

Table 4.6 Resource Utilisation of the proposed architecture vs. other implementations

	Resources	**AWB Module 1 (component)**	**AWB Module 2 (component)**	**Static Part (system)**	**Processor and Mem Interface**	**Total on Chip + 20% of RM**
HLS (Zynq-700) Static Design (2ppc)	**FF**	-	-	16896	PS side	16896
	LUT	-	-	15684	PS side	15684
	BRAM-36	-	-	14	PS side	14
	DSP48E1	-	-	35	PS side	35
HLS (Zynq-700) DPR- low resources utilisation (2ppc)	**FF**	5618	6669	6061	PS side	14063
	LUT	6124	6965	3863	PS side	12221
	BRAM-36	0	0	14	PS side	14
	DSP48E1	13	22	0	PS side	22
Colour image enhancement Virtex Pro-II (1ppc) [147]	**FF**	-	-	29337	-	-
	LUT	-	-	2847	-	-
	BRAM-36	-	-	N/A	-	-
	DSP48E1	-	-	N/A	-	-
Gray block Differencing (1 ppc) IP _ Virtex-5 [145]	**FF**	524		-	-	-
	LUT	839		-	-	-
	BRAM-36	0		-	-	-
	DSP48E1	0		-	-	-
Altera-based implementation (1ppc) [146]	**Logic cells**	2372		16,622	-	-
					-	-
	Memory bits	74,672		794,632	-	-
	M9Ks	12		131	-	-

To compare our DPR approach with other similar methods, the comparison should be applied between similar dynamic approaches. However, no other works have attempted to apply the DPR to such design. Therefore, Table 4.6 shows the resources utilisation of our DPR and static implementations versus other static-based related works. Some of these readings represent the testing system rather than the implemented component. It is clear that the DPR approach can save much of the resources on the FPGA. Some of these implementations proposed simple algorithms that not need much computation effort as claimed by the authors, which means not many resources are needed to be implemented. Other implementations uses software-hardware implementations to reduce the resources such as [108].

4.4.2 Performance Analysis

Due to the adoption of the DPR technique in this implementation and the limitation on the operational frequency of the ICAP port, the design works under two sets of frequencies. The first frequency is the frequency that feeds the configuration engine in the design. This frequency is based on the maximum operational frequency of the ICAP primitive. The ICAP in the Virtex family and the Zynq SoC operates at a maximum frequency of 100 MHz, while it operates at 20 MHz in the Spartan-6 FPGA. The second frequency is the frequency that feeds other components on the design. This frequency varies based on the quality of the implementation and the method.

RTL-Based Implementation: The operational frequency of the AWB cores exceeds 190 MHz using the selected FPGA families. For Virtex-based implementations, a Microblaze soft processor is used to control the system, while an ARM hard processor is used in the Zynq-based implementation. The frequency of the testing platforms dropped down to 161 MHz due to use of other components. Moreover, Virtex-based implementations use NPI memory interface through MPMC to fetch and store data in DDR memory. The Xilinx NPI-MPMC controller can fetch 64 words (double word) every 93 clock cycles, which can limit the performance of system due to this latency at each burst transfer from the DDR memory. On the other hand, Zynq-based implementation uses the embedded memory controller, which has

very good throughput. The system was tested at the aforementioned frequencies and reached a throughput of 322 MP/s if the memory interface performance is neglected and approximately 222 MP/s if the memory interface is involved in the path for the low resource utilisation design. Alternatively, the high throughput rate design reached a throughput of 646 MP/s if the memory interface performance is neglected and approximately 444 MP/s if the memory interface is involved in the path. The core itself without the testing platform can reach higher throughput rates.

HLS-Based Implementation: The operational frequency of the AWB cores is approximately 184 MHz using the Zynq-7000 AP SoC. The frequency of the testing platforms dropped down to 165 MHz due to use of other components. The system was tested at the aforementioned frequencies and reached a throughput of 330 MP/s for the low resource utilisation design. In contrast, the high throughput rate design reached a throughput of 660 MP/s. The core itself without the testing platform can reach higher throughput rates.

To test the performance of the employed configuration engines, three RMs have been generated in different areas in the FPGA (See Figure 4.14). Two partial bitstreams are generated for each RM; the first is generated without compression to obtain the maximum possible file size, whereas the second is for a black-box of the same area as the RM. The first RM is generated with only CLB components. The second contains BRAM and CLB components while the third one contains CLB, BRAM, and DSP components. The black-box is generated with compression enabled to obtain the smallest file size. Both configuration engines adopted in the above designs can utilise the maximum speed of the ICAP primitive, but due to the use of NPI memory interface, the average throughput can drop down compared to the non-NPI memory interface method. Table 4.7 shows the configuration times for the benchmark RMs using the Zynq-based configuration engine that employs the PS hard memory controller and the Virtex-6-based configuration engine that uses the NPI-MPMC controller when operating at 100 MHz and employing the 32-bit configuration of the ICAP.

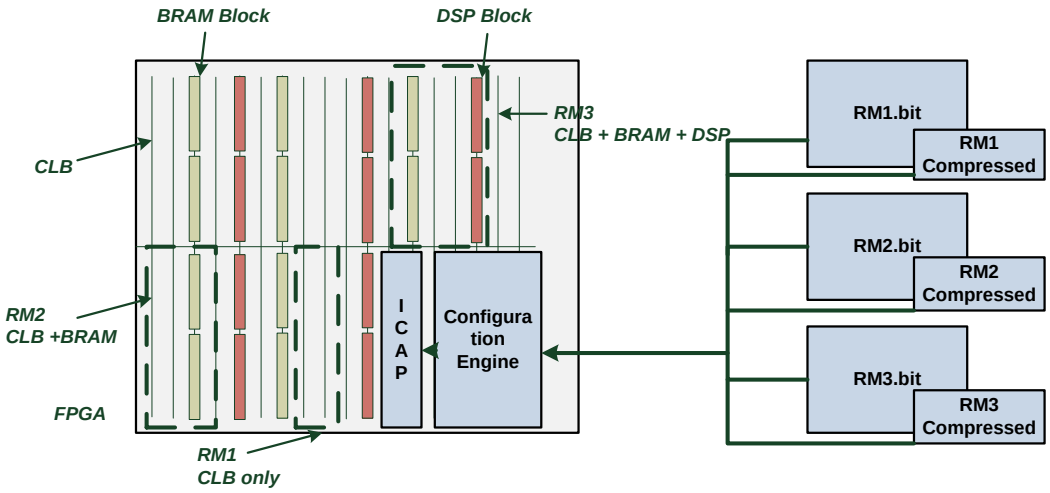


Figure 4.14 Configuration engine evaluation

Table 4.7 Configuration Times of the Proposed Configuration Engines

RM ID	Zynq-7000 configuration engine				Virtex-6 NPI-MPMC Controller (DDR3) configuration engine			
	Configuration Time (us)				Configuration Time (us)			
	Normal	RM size (KB)	Black-Box	RM size (KB)	Normal	RM size (KB)	Black-Box	RM size (KB)
RM1	305.1	120	98.4	38	45.38	12.3	16.64	4.5
RM2	496.7	196	168.92	66	268.45	73.5	70.14	19
RM3	633.2	250	272.48	107	386.8	106	95.75	26
Avg. Throughput	385.01 (MB/s)	-	380.72 (MB/s)	-	267.45 (MB/s)	-	264.04 (MB/s)	-

The bitstream of each module should be extracted through the Xilinx and Vivado DPR flow. The size of each module depends on the size of the allocated reconfigurable region and the amount of utilised resources. In most FPGAs, the tool enables the designer to generate a compressed version of the bitstream to get the smallest size, as mentioned previously. The compression ratio is inversely proportional to the resource utilisation in the allocated reconfigurable region. If the module uses more than 50% of the resources in the region, there is no chance to get a high compression ratio compared to the black-box, which is an empty module. The

average throughput of the implemented configuration engine in Table 5.7 is comparable to the results obtained from other proposed configuration engines in the literature such as [45] [46] and [47].

Table 4.8 shows the size of the partial bitstreams of the DPR designs and the configuration time for each bitstream if the memory interface timing is involved or not involved in the case of virtex-6. Moreover, it shows the configuration time for the compressed bitstream version. The compressed bitstream size is based on the minimum compress ratio between the two modules.

Table 4.8 Size of the Partial Bitstreams and Their Configuration Times

Implementation	*Size of bitstream (KB)*		*Configuration time (ms)*		
	Virtex-6	*7-series*	*NPI-MPMC interface Virtex-6*	*NPI-MPMC not involved - Virtex-6*	*DMA 7-series FPGA*
DPR- low resources utilisation	461 KB	436 KB	1.72	1.18	1.1
Compressed	324 KB	302 KB	1.21	0.83	0.77
DPR- high throughput rate	795 KB	581 KB	2.96	2.04	1.47
Compressed	586 KB	401 KB	2.19	1.50	1.02

The system is designed to process a range of image sizes, but for testing purposes, images of size 1920x1080 pixels have been used to analyse the performance. The images are fetched either from the camera sensors or from the DDR memory. Since the algorithm needs to go through the input images twice—the first time to get the parameters and the second to correct the pixel values—the latency of the system is equal to the time of the first scan plus the time of configuring the second partial module plus its logic time. The latency of the DPR designs and the static design is shown in detail in Table 4.9.

The total time for executing and processing one image of size 1920x1080 pixels equals the time of scanning the image by the algorithm twice plus the time of configuring the partial module for DPR implementation, while it needs only two

scans for the static design. The static design was tested and performed better than the low resource utilisation DPR technique, but not better than the high throughput DPR implementation due to the increase in the design parallelism while maintaining the resource utilisation in the range of the static design. The total time for executing one image using the static design is approximately 18.6 ms. Table 4.9 shows the executing time for one image employing the three different designs using the RTL- and HLS-based approaches. The above timings are the timings for different FPGA families, as all of them use the same configuration frequency except for Spartan-6, as mentioned previously.

Table 4.9 Execution Time and Latency for the Proposed Implementations

Implementation	Stage	Static design		DPR- low resources utilisation		DPR - high throughput rate	
		Cycles	Time (ms)	Cycles	Time (ms)	Cycles	Time (ms)
RTL (Virtex-6) NPI-MPMC memory interface	**First scan**	1506600	9.32	1506600	9.32	753300	4.66
	Config. time 100 MHz - not compressed	-	-	171492	1.72	295530	2.96
	Second scan	1506600	9.32	1506600	9.32	753300	4.66
	Logic time	15	≈ 0	15	≈ 0	15	≈ 0
	Latency	**1506615**	**9.32**	**1678107**	**10.97**	**1048830**	**7.62**
	Total	**3013215**	**18.6**	**3184707**	**20.36**	**1802130**	**12.28**
HLS & RTL (165 MHZ) (Zynq7000) PS Embedded memory interface and processor	**First scan**	1036800	6.28	1036800	6.28	518400	3.14
	Config. time 100 MHz - not compressed	-	-	110392	1.1	146958	1.47
	Second scan	1036800	6.28	1036800	6.28	518400	3.14
	Logic time	15	≈ 0	15	≈ 0	15	≈ 0
	Latency	**1036800**	**6.28**	**1154692**	**7.38**	**684358**	**4.61**
	Total	**2073615**	**12.56**	**2191507**	**13.66**	**1183773**	**7.75**

Similar works are presented in the literature, but none of them uses the DPR feature to make the comparison consistent. In [108], a hardware-software co-design has been implemented for AWB on Virtex-4. The minimum execution time achieved was 30 ms for an image of size 320x240. It is illogical to compare the two works because they use two different methods and technologies. Moreover, the AWB core part of the work in [105] was implemented on the processor side, which again cannot be compared to our work. In [102], a static design for the gray edge hypothesis was implemented, but no specific information about the performance or resources was provided. Table 4.10 compare between the implemented DPR design and the available implementations in the literature in terms of execution time, frequency, and throughput. In our implementation an image of size 1920x1080 is used. The amount of Parallelism in our design gives the architecture additional throughput. However, in implementations 1 and 3, the frequency is better than ours due to the use of simpler approaches and algorithms. Some of the presented implementations are based on non-learning approaches unlike our approach, which applies parameters based on statistical information from the image itself. The non-learning approach just applies pre-known parameters, which increase the overall performance.

Table 4.10 Performance comparison between the proposed design and similar designs

Implementation	Execution Time (ms)	Throughput (Mpixel/s)	Frequency (MHz)
Our DPR implementation (4ppc)	7.75	268 (per 2-scans on image)	165
Our Static implementation (2ppc)	13.82	150 (per 2-scans on image)	165
Implementation 1 [147]	8.6 (1600*1200)	222 M Sample/s	224
Implementation 2 [146]	Not mentioned	130	Not mentioned
Implementation 3 [145]	-	-	333.3

4.4.3 Power Consumption Analysis

This section analyses the power consumption of the implemented designs. Just few works in the literature discusses the power consumption of such designs with similar functionality, so this section discusses deeply the impact of the DPR feature on power in the implemented designs in addition to the comparison with other similar designs. The following methodology is used to evaluate the power consumption of static implementation versus the DPR implementation. In this work, similar to the previous chapter, the power consumption has been measured using two different methods to verify the overall results. In the first method, the power information is read directly while the system is working. In this method, the system monitor and ChipScope Pro tool are used to access the information via the JTAG in Vertix-6 [135]. In the Zynq-7000 AP SoC, the TI Fusion Power Designer tool is used to monitor the power information on the Zynq board [136] (See Figure 3.12, chapter 3). This tool communicates with the embedded power regulators and PMBus controller inside the Zynq board over the serial bus to fetch the power information or control it (See Figure 3.13, chapter 3). The second method extracts the power information from the output file of the processed design using power analysis tools such as Xilinx Xpower tool [137] for Virtex-6 and the power analysis tool part of the Vivado suite for 7 series FPGAs [138]. After obtaining the power readings via the above methods, a comparison is made to find the impact of using DPR on implementing the system.

In Zynq-based implementations, the design is split into two parts between the PS and PL sides. The PS side consumes more power compared to the PL in this implementation for two reasons. First, the clock resources are generated in the PS side, and second, the DDR memory is attached directly to the PS side. The PL side contains the other components, which are mainly logic components such as FFs, LUTs, and DSPs. For DPR implementation, Figure 4.15 shows the power dissipation of the internal logic circuit of the PL side of the Zynq board for the proposed DPR HLS-based implementation using the Fusion power designer tool. The sudden power increases in the figure show the power consumption when the design is processing data, while the rest shows the consumption in the idle state. Due to the low polling

rate in the tool, the reading cannot be obtained in the normal case. Each module was repeated 1000 times to obtain these readings. The figure is divided into five sections based on the running module. The first section of the figure shows the power consumption when the design uses a black-box module instead of AWB module. The second section shows the total power consumption when the first module is configured. The third section shows the power consumption when the system is at idle state and in the existence of the first module. The fourth section shows the system power consumption when the second module is configured. Finally, the fifth section shows the power consumption at idle state when the second module is configured. It is clear that the power was dropped by configuring the black-box module due to absence of logic components, which led to low dynamic power consumption. On the other hand, Figure 4.16 shows the power consumption of the internal logic circuit of the PL side for the static design. The figure shows the difference in power consumption compared to the DPR implementation due to the existence of two modules on the FPGA at the same time, which means more logic activities and more logic leakages thus more power consumption. At idle state, the static design consumes approximately 220 mW, while the DPR design consumes only 140 mW if a black-box is configured or 160 mW if any module is configured. Moreover, the DPR design consumes approximately 230-250 mW at the time of processing data, which is much less than the 300-320 mW consumed by the static design at the same state. The 20 mW difference shown in the two figures is due to the difference in the activities in each module and the amount of components used. The power consumption of the PS side in static and DPR designs are equal. Figure 4.17 shows the power consumption of the PS internal logic circuit including AXI buses, I/O, and the processor logic, which is approximately 340 mW with possible +60 mW based on the processor activities. The DDR and clock generator consume around 889 mW and 350 mW, respectively. To reduce the power of the PS side, the clock can be reduced to a minimum level at idle states to reduce the overall power consumption where it is required only to generate high frequencies.

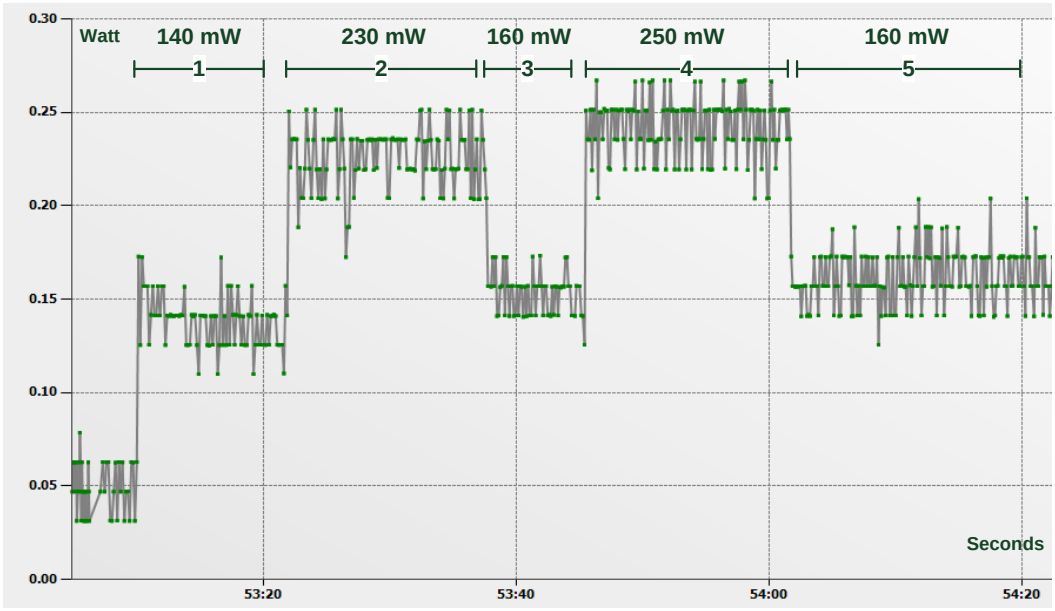


Figure 4.15 Power consumption of the PL side for the proposed DPR HLS-based implementation on Zynq board (2 ppc)

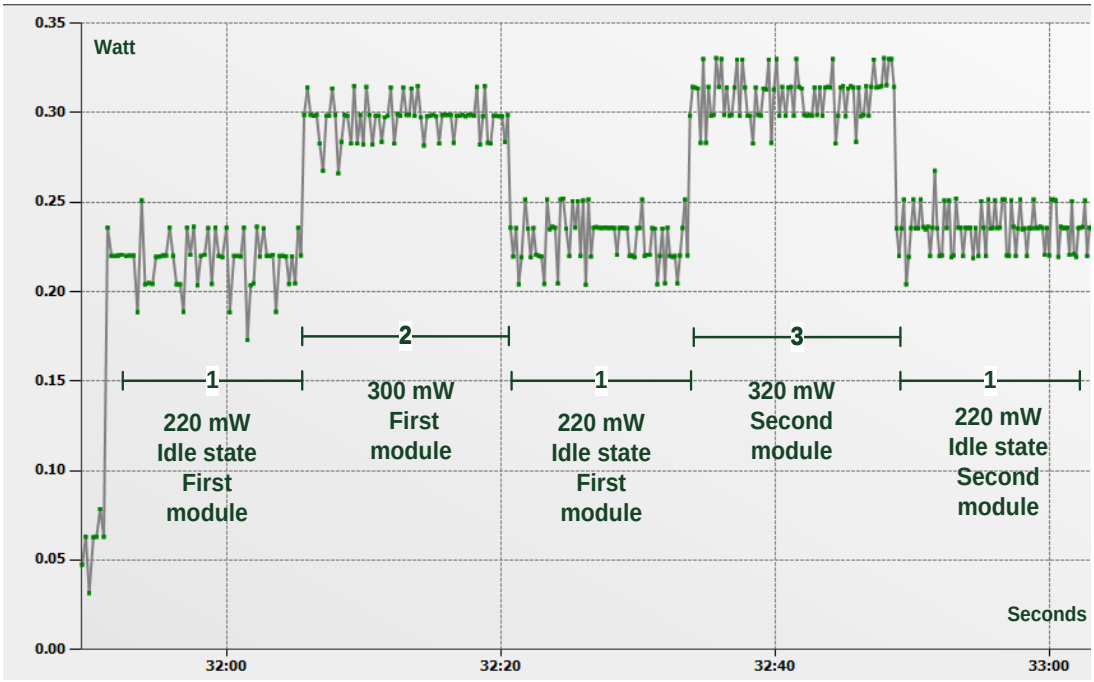


Figure 4.16 Power consumption of the PL side for the proposed static HLS-based implementation on Zynq board (2 ppc)

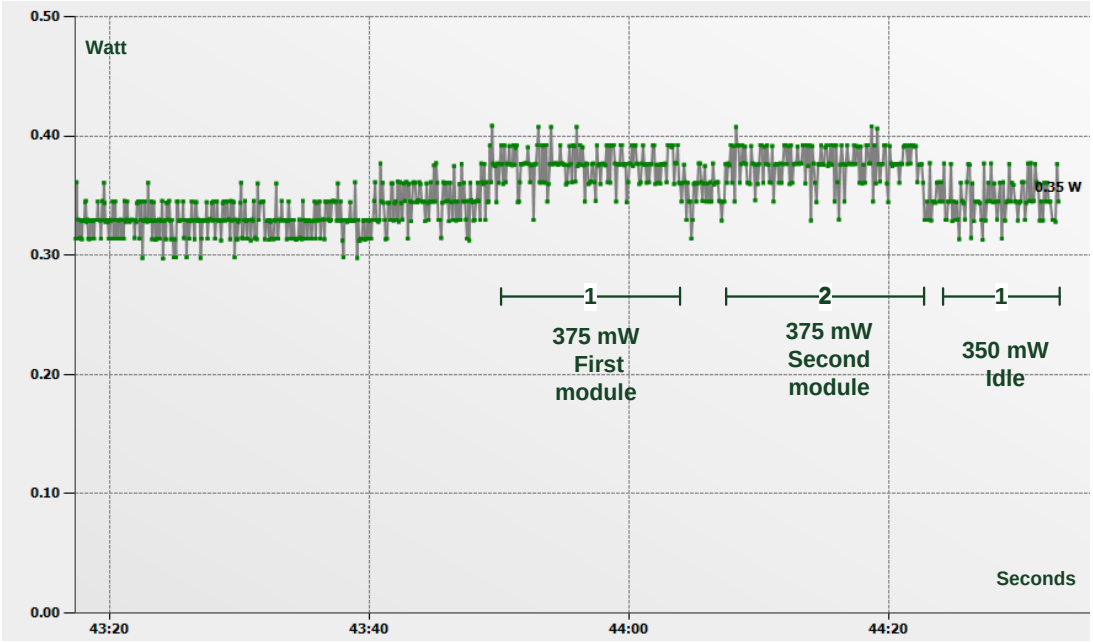


Figure 4.17 Power consumption of the PS side for the proposed static and DPR HLS-based implementation on Zynq board

Similar results have been obtained through using the Xilinx power analysis and Xpower estimator tools. Unlike the previous tool, these tools have no information about the power at each stage of the design or at the idle states. Table 4.11 and Table 4.12 show the power consumption of the DPR and the static designs and the amount of saved power gained by adopting the DPR technique on number of FPGA families. The values recorded in the tables show the total power, including dynamic and static power. The dynamic power varies based on the design activity (see Figures 4.15 and 4.16).

Table 4.11 Power Consumption of the DPR AWB Implementation

Chip	**Dynamic Power Consumption (mW) (including static power)**		
	Static + PR module B	***Static + PR module A***	***Static + black box***
Virtex 4	80	79	75
Virtex 5	350	347	311
Virtex 6	370 (3880)	368 (3878)	340 (3850)
Zynq - RTL	141 (296)	152 (310)	90 (244)
Zynq - HLS	150 (309)	161 (320)	95 (254)

Table 4.12 Power Saving in DPR vs. Static Implementation

Chip	Static design *Dynamic Power Consumption (mW) (including static)*	Saving in DPR
Virtex 4	91	13 %
Virtex 5	391	11 %
Virtex 6	408	10 %
Zynq - RTL	291 (451)	47 %
Zynq - HLS	301 (465)	44 %

Based on the above tables and figures, the total power consumption appearing in the figures is less than the recorded values in the table. This is because the values appearing in the table are based on the maximum and total activities for all components on the design, while the value in the figure is the value for the current working components only. Moreover, the high saving in DPR Zynq implementations is due to the high footprint of the RM module in the full PL design compared to the other implementations on other boards where the design includes soft processor and memory controller in the PL side. Figure 4.18 shows the total power consumption for DPR and static implementations on the Zynq board and the power consumed by the implemented cores.

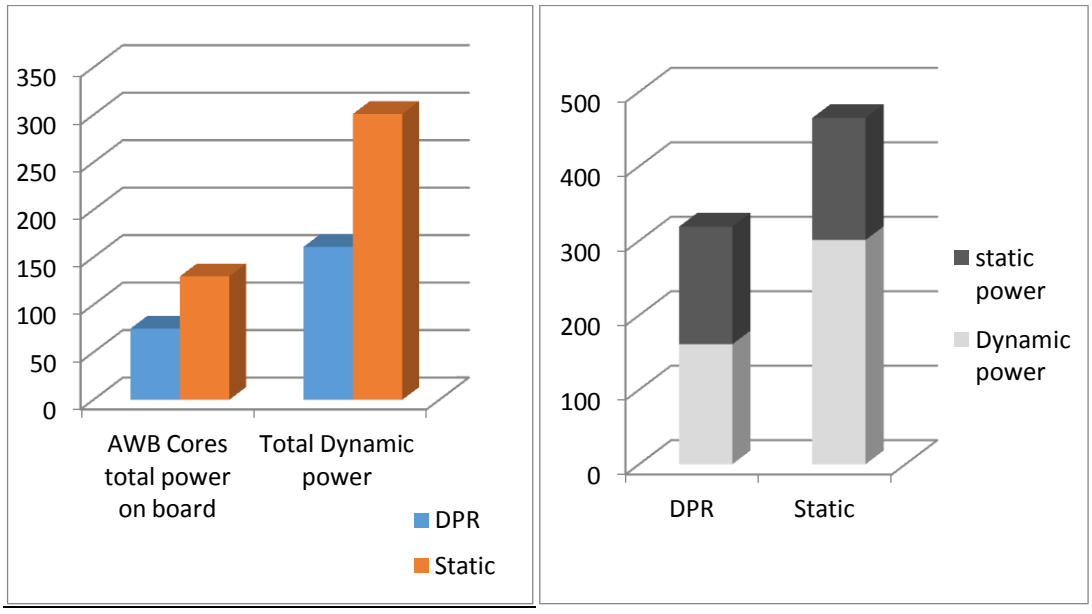
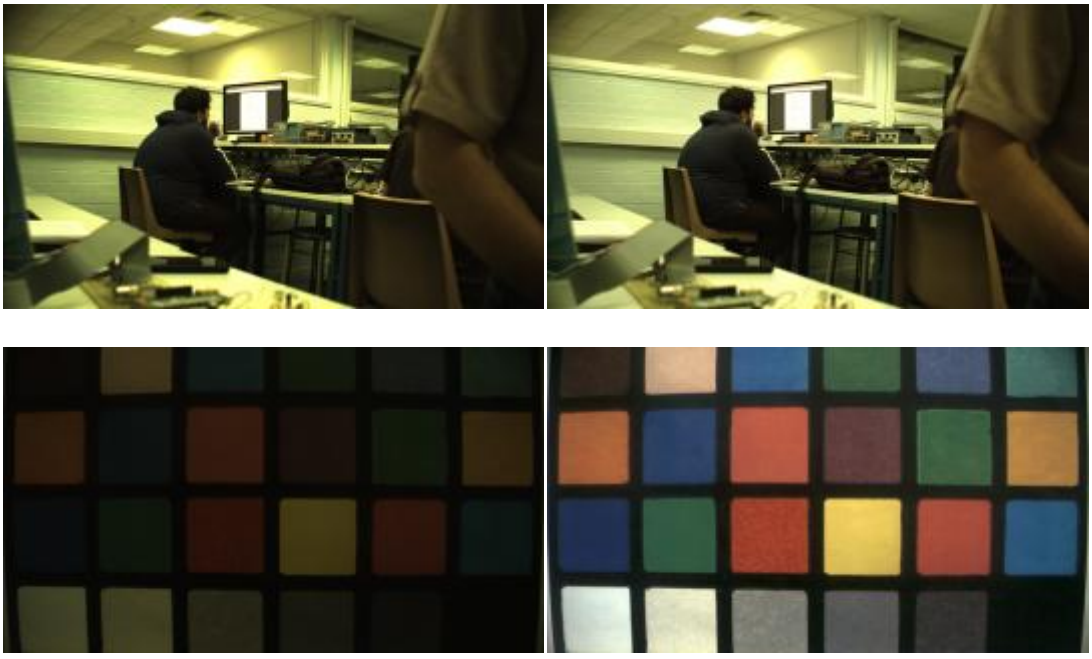


Figure 4.18 PL-power consumption of DPR and static-based implementations on Zynq-7000 AP SoC

On the other hand, few works in the literature investigate the power dissipation of such function. In [150], an AWB engine has been build part of image sensor pipeline. The engine is one of eight engines in the pipeline. The complexity of these engines varies based on the amount of computation in each engine. The work showed that the ISP block consumes 578 mW. This is the total power dissipation of the static ISP style implementation. In our design, the same ISP can be implemented with an average of 320 mW of power dissipation. This shows how the DPR can decrease the total power dissipation.

4.4.4 Output Images

Figure 4.19 shows the images before and after applying the AWB algorithm using the implemented designs, no matter if DPR or static design is used. As mentioned previously, for testing purposes the AWB algorithm was applied to the interpolated images to recognise the changes at the processed images, as it is originally applied to CFA patterned images.



(a) **(b)**

Figure 4.19 (a) Unprocessed images (b) Processed images using AWB

4.5 Summary

Most image processing algorithms need high computational power to process the data in real time. Moreover, the complexity of these algorithms leads to big cores that use a large footprint of the available chip area. Combining the number of image processing cores to build an imaging system could cost a large amount of resources and consume much power.

This chapter presented a novel implementation of one of the AWB algorithms that processes the images to keep the colour of objects constant under different illumination conditions. The architecture exploits the parallelism and DPR features provided by FPGAs to increase the performance of the algorithm, reduce resource utilisation, and decrease power consumption to meet the processing requirements of modern cameras. The algorithm was implemented using two different approaches: static implementation and DPR implementation. A comparison between the experimental results of both approaches has been carried out. The comparison showed that the DPR implementation performs better than the static in all three aspects for the selected algorithm. DPR implementation showed around 40%, 30%, and 35% reduction in power, speed, and resource utilisation, respectively. In DPR implementation, the use of a high-throughput internal configuration port can increase the throughput of the implemented design. An enhanced configuration engine was implemented to utilise the full speed of the available ICAP primitive.

Chapter 5 : A Dynamic Partial Reconfiguration Architecture for Cameras and Imaging Systems on FPGAs

With the increasing number of smart devices being used in our daily life, capturing our life moments has been becoming an essential part of our lifestyle. Therefore, most companies try to embed a digital camera in their smart devices. The image processing systems in general require a high computation power; therefore, the structure of these systems should fulfil specific requirements of power consumption, performance, and throughput to be a valid image processing system. Traditionally, these systems are built using ASICs due their low power consumption, high-speed optimisation, and low unit cost. Today, FPGAs appear to be a competitive environment in this field. Modern FPGAs have a number of features, such as low power consumption compared to the old generations, high speed, fast time to market, and the most important feature the flexibility, which could make them a good choice for such systems. The term flexibility refers to a number of aspects in FPGAs; the primary aspect is the reprogrammability [151], which enables the user to adapt to changing standards and support design reuse. Moreover, FPGAs provide additional flexibility to the designers by using the DPR feature, which exists in modern FPGAs (see chapter 2).

This chapter presents an implementation for an FPGA-based image processor that uses the DPR feature to enable the employment of unlimited number of imaging enhancement stages. Moreover, it shows how the implementation utilises resources and reduces power consumption. The chapter shows an enhanced version of the system, which uses a pre-fetching feature to increase the performance. The proposed implementation can be customised to target different imaging applications. A camera

implementation is used as example in many places in this chapter. Finally, the chapter presents a case study for the system alongside the proposed R4THOS to enable the idea of autonomous car.

5.1 Background and Related Work

5.1.1 Related Works

Digital cameras internal components can be categorised into three main types: the input, the processing, and the storage or output components. The input part captures the image data using sensors. The processing component performs the baseline and enhanced image processing on the raw data produced by the camera sensors to make them look like what we see using our eyes. This part is usually called image-processing pipeline. The final part is the storage and output part, which either saves the result in non-volatile memory or shows it on the screen. Obviously, the image-processing pipeline is the core part of any digital camera and decides the quality of its images. Moreover, it requires a significant amount of computation power because it consists of a number of complex algorithms working sequentially. The critical issue in building this part is the timing demands. Therefore, it is important to be built in a high-performance environment such as ASICs, DSPs, or FPGAs, but not using GPPs (see chapter 2). Many research studies have discussed this field, but most of them focused on the traditional way of building such systems. In [72], Texas Instruments developed a full commercial imaging System on Chip (SoC), which consists of an ARM processor, DSPs, and application specific hardware. The processing pipeline was implemented in a configurable manner to combine the performance advantage from the hardware with flexible control to fine-tuning of the algorithm parameters. Moreover, the majority of modern digital cameras have ASIC-based media processors with all functions required for processing the data from the sensors to storage. All these functions are embedded in the chip but most imaging companies try to hide the architecture of their own processors and the algorithms used in the individual components for competitive purposes. The following are examples of such media processors: Canon have their DIGIC series while Nikon Company uses its own EXPEED media processor. The EXILIM engine is used in

Casio cameras. These architectures are not flexible so it cannot be changed after fabrication, which reduce much of their flexibility.

With the appearance of modern FPGA technology, a number of research studies have discussed developing such systems on FPGAs. Moreover, many leading companies started to test and develop such systems on FPGAs to evaluate their performance and efficiency compared to the traditional methods. Xilinx has built an IPP [105] on the 7000 all programmable SoC in a configurable manner to adjust the algorithm parameters on the fly, either manually by the user or automatically, based on statistical information gathered from the frames data. Altera, the FPGA leading company has built their own vision system to perform complex inspection tasks in addition to digital image acquisition and analysis [150]. The resulting data can be used later on pattern recognition, object sorting, and tracking. In [104], a camera image signal processing core has been implemented to be used in different cameras. Generally, this type of static architecture has been proposed in different research works with different properties. On the other hand, different approach has been discussed in other research works such as [115], which proposed a Coarse grained reconfigurable image stream processor that consists of different elements, each of them is specialised in a specific image processing functionality. The architecture divides the imaging pipeline into sub-pipelines configured one after the other. It uses these elements to build up each sub-pipeline. In addition, the research works [152, 153] present an FPGA-based flexible smart camera based on DPR. The idea is based on loading pre-defined modules in specific region in the FPGA with the aid of a communication infrastructure to connect the loaded modules. Other FPGA-based implementations are presented in chapter 3. Most of these implementations focus on the performance. In this work, we look at the performance as well as the flexibility, area utilisation, and power consumption to increase the system efficiency.

5.1.2 IPP

The data captured by the camera sensors should match the scene seen by our eyes. Unfortunately, this does not happen. Therefore, a number of image enhancement and

constructing stages are required to adjust the scene data to look similar to the scene seen by the eyes. In this context, different companies use their own image-processing pipeline that uses customised algorithms in each stage. Figure 2.23, chapter 2 shows the IPP based on Davinci technology proposed by Texas Instruments. All the processing components need an intensive computation power to meet the real-time processing requirements. No perfectly typical image-processing pipeline exists, but in general, anyone should include a minimum number of stages that adjust the scene to look similar to the scene seen by the eyes. These stages follow:

- **Colour Interpolation**: The colour interpolation stage is needed to construct the full image and predict the missing colours in each pixel because the overlaid sensors pass only one colour at each pixel. Many algorithms have been proposed for this stage (see chapters 2 and 3).

- **AWB**: Due to different illumination conditions, the colour of objects varies based on the temperature of the illuminated light source. In this stage, the colour is kept constant under any light source by calculating parameters extracted from the image data (see chapters 2 and 4).

- **Gamma Correction:** Adjustments applied to the image when it is displayed in the screens due to the difference in colour representation in monitors.

- **Other Stages:** Includes noise reduction, space conversion, edge enhancement, and contrast enhancement.

- **Post-Processing**: An example is compression, which is used to reduce the total image data for storage purposes. Formats include JPEG or JPEG2000. It could be a lossy or lossless standard.

5.2 DPR Image Processor

The DPR image processor is a unique architecture that allows the user to process different imaging functions dynamically by swapping tasks in and out to provide more flexibility compared to current state of art processors. The repetitive nature of

image processing tasks increases the efficiency of this approach. The implementation of the proposed imaging system is illustrated in Figure 5.1. The part used for implementing the system is shown in Figure 5.2. The design exploits the DPR feature in modern FPGAs to reduce the resources and power consumption by configuring tasks internally through the internal configuration port ICAP using the bitstream saved in the DDR memory. The idea is to keep the minimum number of processing elements on the chip at a time. This practice will reduce the total area utilisation as well as the power consumption, which makes it possible to implement complicated imaging applications in small and limited resources FPGAs. The implementation can be used as a template to process unlimited imaging tasks with no specific order. Moreover, the flexibility of the system allows the user to change standards easily by swapping their corresponding bitstreams in and out internally through the ICAP. In addition, the system can be customised towards specific imaging application dynamically.

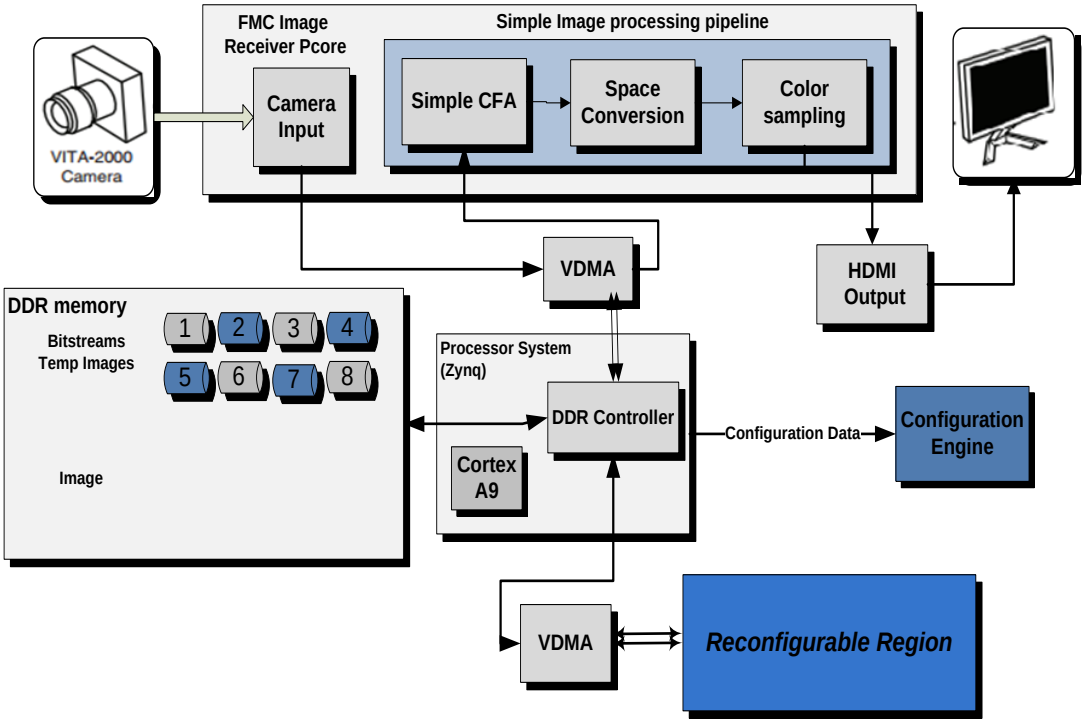


Figure 5.1 Imaging system block diagram

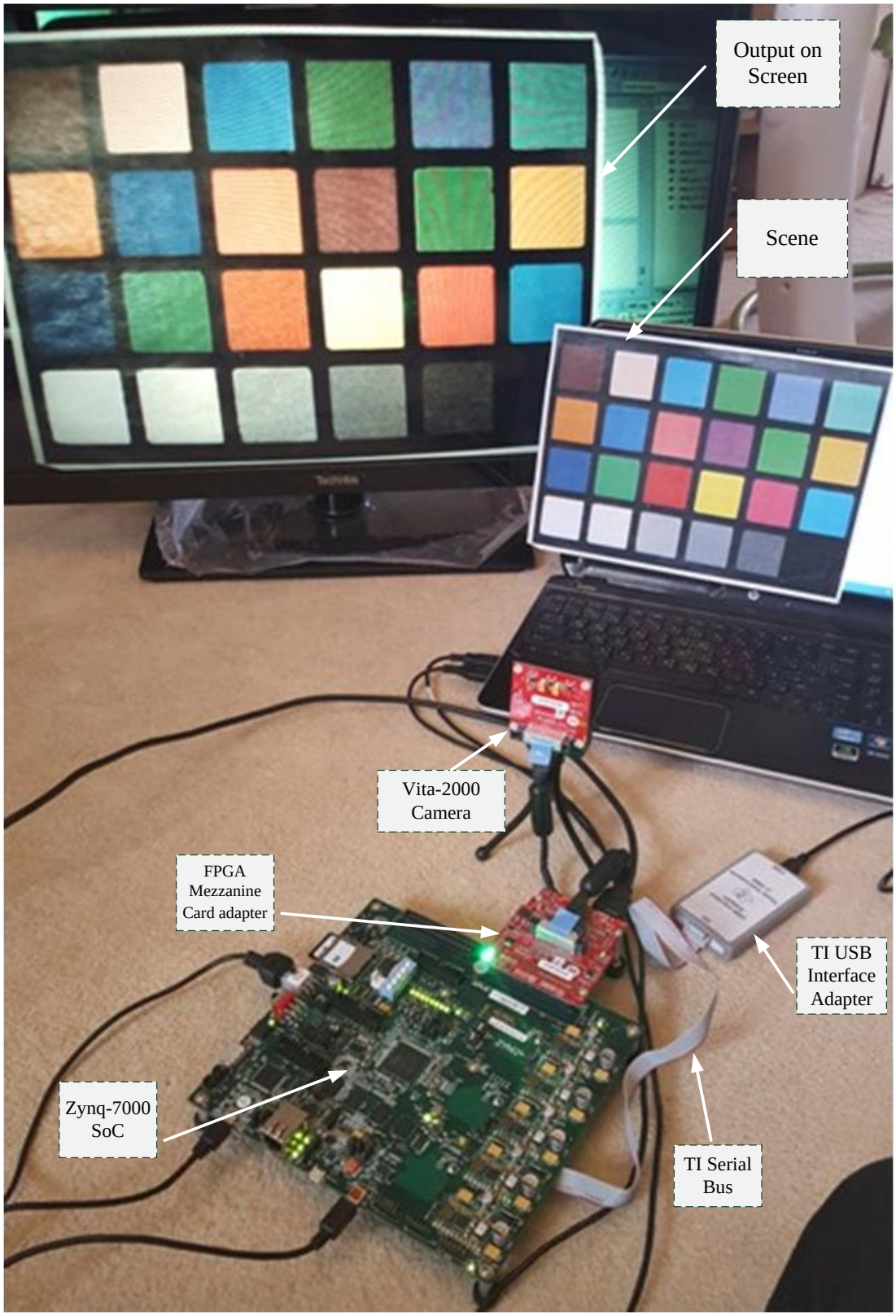


Figure 5.2 Imaging system parts

5.2.1 Implementation

The implemented imaging system is divided into two main parts, as shown in Figure 5.1: the static part and the reconfigurable part. The static part contains a simple processing pipeline primarily to allow the user to capture a specific scene. In addition, it contains the FMC-IMAGEON VITA receiver core [154], [155], the HDMI output interface, and the IPs, which are used in data communication between the modules and memory components such as VDMA. On the other hand, the reconfigurable part contains the aspect of the design that can be replaced regularly to change the system functionality. In this case, these are the image-processing pipeline stages mentioned previously. The implementation works as a regular camera in which the image processor is the reconfigurable part and static part represents the other components of the camera. In this context, all the data are passed through stages of the processor sequentially until the last stage. The implemented DPR image processor can conserve much power by keeping only one stage active at a time.

- **Implementation steps and tools:**

The system has been implemented using Xilinx tools as follows:

- **Vivado Integrated Design Environment (IDE):** Part of the Vivado design suite. It is the new generation of Xilinx development tool. It replaces the Xilinx ISE and Xilinx Platform Studio (XPS). It is mainly used as a solution for designing embedded processing systems. Moreover, it is used in synthesis and analysis of HDL designs, to perform timing analysis and examine RTL designs (see chapter 2).

- **Vivado HLS:** The tool accelerates IP creation by enabling C, C++, and system C to be targeted into Xilinx devices without the need to create RTL manually using HDL. This tool has been used to develop the IPP components. (see chapter 2).

- **Software Development Kit (SDK):** The tool is used as an environment for creating embedded applications on the embedded processors such the ones

existing in the Zynq SOC and MicroBlaze, a leading soft processor. The C-language software is used to control the design and the DPR swapping process.

The following steps are used to develop the system and are based on the aforementioned tools:

- Develop the IPP components in C language individually and then let the Vivado HLS converts the codes to Xilinx IPs that can be used directly in the system.
- Develop the full imaging system using Vivado IDE. Only one component of the image-processing pipeline is included as part of the design.
- Create C-application for the embedded design using the drivers provided by each IP in the design. The application should use Vivado SDK to load the code on the embedded ARM processor.
- Apply dynamic partial reconfiguration on the part that holds the IPP component. The process should follow the DRP design flow explained in section 2.1.2.1 to generate the partial bitstream for each RM.

➢ **Static Part:**

In the static part, the simple processing pipeline consists of colour filter array interpolation, colour space converter, and Chroma resampler Xilinx IP cores. The cores are used only for showing the data received by the design in the output side to let the user capture the desired scene using the screen. Also in the static part, the FMC IMAGEON VITA receiver core controls and synchronises the stream of data from the image sensors to the user design. The HDMI output core controls the flow of data with the HDMI interface. The VDMA is used to redirect the streamed data to the DDR memory and return them back to the design. This action allows the user to process any frame of the data once the user presses the capture button. At that moment, the system will keep the last frame in the buffer, which is allocated in the DDR memory side. All static part components are synchronised using Xilinx Video Timing Controller (VTC) core to keep the flow of data smooth between any two adjacent components.

- **Reconfigurable Part:**

The reconfigurable part consists of the IPP stage or the imaging task if the system is customised to target different imaging application. Only one stage exists in the design at a time. The stages are swapped sequentially using the DPR feature. At the end of the previous task, the done signal triggers the system controller to release the current task and fetch the next task. To configure the stage dynamically, the current configured stage should be completely isolated from the system before configuring the new stage; otherwise, an unknown data will be sent to/from the reconfigurable region during the configuration process, which results in halting the entire system. The user should reset the entire RM and disconnect any data bus that is connected to the RM—specifically, the AXI bus. All these activities should be controlled by the user application each time a new task is configured. When the configuration process is over, the controller will release isolation signals that isolate the RM from other parts on the system.

Three stages of the image-processing pipeline have been implemented using the Vivado HLS tool: demosaicing, AWB, and noise reduction. The demosaicing stage has been implemented based on the Adams-Hamilton interpolation algorithm (see chapter 3). The AWB stage is based on gray world and retinex theory (see chapter 4). The implementation is divided into two stages to reduce the resource utilisation and power dissipation. The noise reduction stage uses a 3x3 median filter. All stages are optimised to be able to work under a frequency of 160 MHz. This frequency allows the system to process HD images with a rate of 77 frames per second if one pixel per clock cycle is processed. The rate can be increased by increasing the number of processed pixels per clock cycle. Figure 5.3 shows the system data flow chart. It illustrates how the system data are passed from sensors to memory and to the reconfigurable area. The system is designed in a way that the data are passed through a buffer in DDR memory as a constant stream of data from the sensors to the system logic. This practice will keep a number of frames of the video stream available any time on the memory. Once the capture scene command is issued, the system stops streaming data through DDR and keeps the last scene in the DDR memory frame

buffer to be used in processing the scene by the IPP processing elements, one after the other. Once the task is completed, the next task is configured by the system, and the data are brought again from the buffer to be processed and returned back to the buffer.

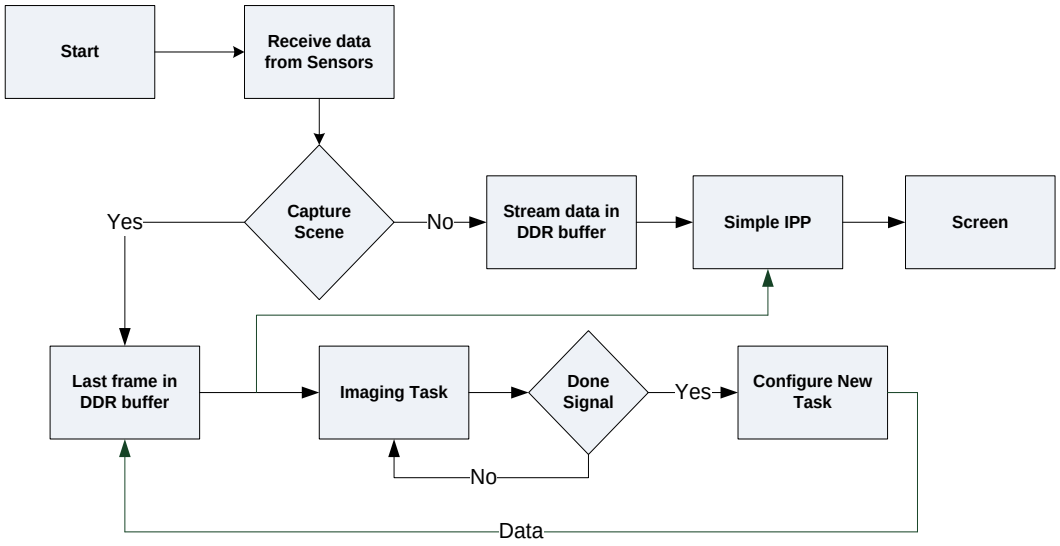


Figure 5.3 Data flow of the DPR imaging system

- **Task Configuration:**

In Zynq boards, the PCAP, which resides in the processing system side, is used to configure full or partial bitstreams. The port is the default configuration port in Zynq board. To configure the PL side using other ports, the PCAP port should be disconnected first before move the control to the other port such as the ICAP. The PCAP configuration path is shown in Figure 5.4. Although the configuration primitive provided in the FPGA can support up to 400 MB/s configuration throughput, the transfer rate through the PCAP is approximately 145 MB/s [11]. The overall throughput is limited by the PS AXI interconnect. Therefore, this limitation could cause some problems for complex designs, which have large reconfigurable regions that need frequent tasks swapping, which will consume much time for configuration only. To overcome this limitation, a fast configuration engine has been designed and deployed to increase the configuration speed and to utilise the full speed of the ICAP primitive. The configuration engine consists of a DMA to move

data from the memory to the ICAP primitive in addition to the ICAP interface. The ICAP interface is a direct interface. The valid data signal is connected to the ICAP enable signal in a write mode, as the new generation of ICAP in 7-series devices has no busy signal [156]. Figure 5.5 shows the designed configuration engine diagram. The DMA IP is configured with the max burst option to increase the overall throughput. The configuration engine is only 847 LUT and utilises only 1.5% of the programmable logic in Zynq 7020 SoC. The configuration engine is explained in detail in section 4.3.1.

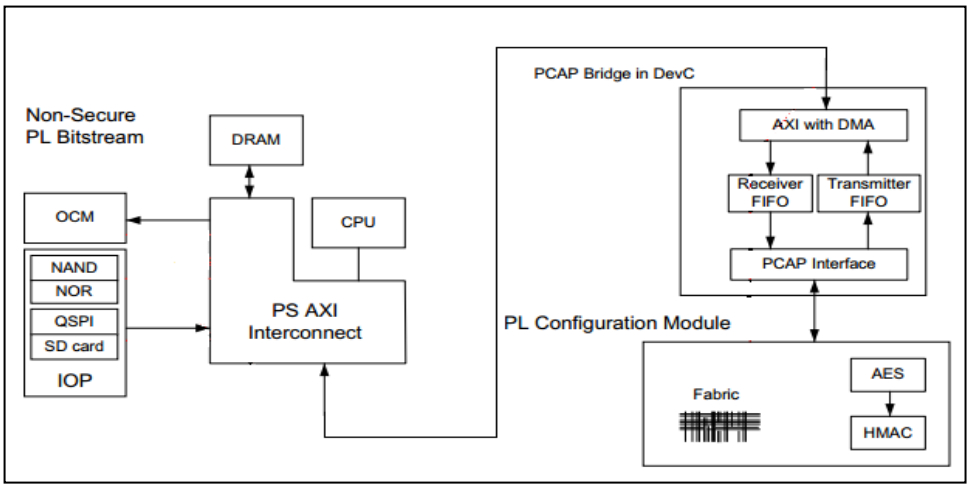


Figure 5.4 PCAP configuration path [11]

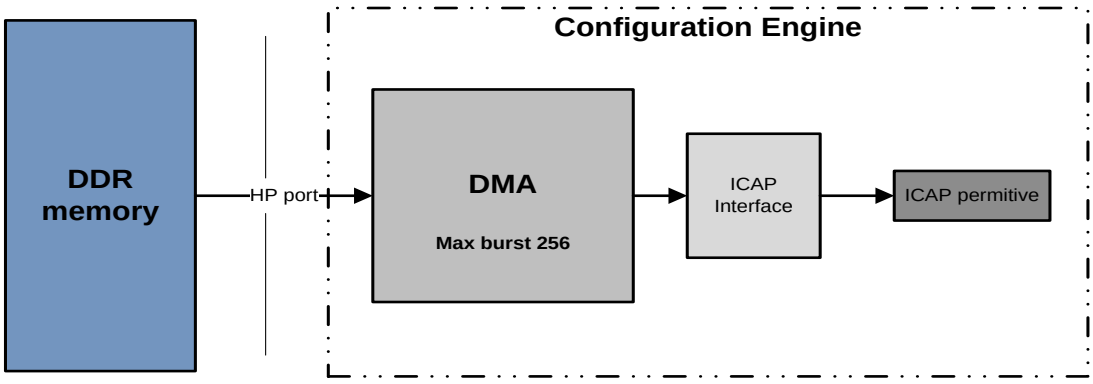


Figure 5.5 The deployed configuration engine

5.2.2 Experimental Results

The system has been implemented on the Zynq-7020 SoC evaluation board. The system has three different frequencies: 200 MHz for the IMAGEON VITA receiver

core, 150 MHz for the image-processing pipeline, and 100 MHz for the configuration engine, as this is the maximum operational frequency for the ICAP primitive. Moreover, the 100 MHz is used to synchronise the AXI control buses with the processor. Two different sets of RMs have been implemented; the first set processes one pixel per clock cycle to get a throughput around 150 MPixels/s. The second set of modules processes two pixels per clock cycles, which leads to achieving a double throughput compared to the first set.

- **Performance and Resource Utilisation:**

The reconfigurable region size depends on the resources utilised by the largest stage among the image-processing pipeline stages. In this design, the reconfigurable region is approximately 15% of the total chip area if the first set of modules is used. Figure 5.6 shows the floor planning of the implemented system, and Table 5.1 shows the resource utilisation for different IPP stages for the two sets of modules, set one refers to 1 pixel per clock cycles and set 2 refers to 2 pixels per clock cycle. The system has been tested with only three different stages. The expirements show that the reconfigurable region should have 20% more resources compared to the largest stage for routing purposes. All the stages have been implemented using Vivado HLS tool.

Table 5.1 Resource Utilisation of RMs

IPP stage	Resources (set 1/set 2)			
	LUT (1/2)	***RAMB18 (1/2)***	***DSP48 (1/2)***	***Slices (1/2)***
Colour Filter array Interpolation + Median Filter	5049 / 6320	23 / 23	0 / 0	1312 / 1630
Automatoc White Balance part1	4030 / 6124	0 / 0	10/ 13	1057 / 1581
Automatoc White Balance part2	5302 / 6669	0 / 0	12/ 22	1380 / 1667
Reconfigurable Region	6400 / 8000	40 / 40	20 / 40	1600 / 2000

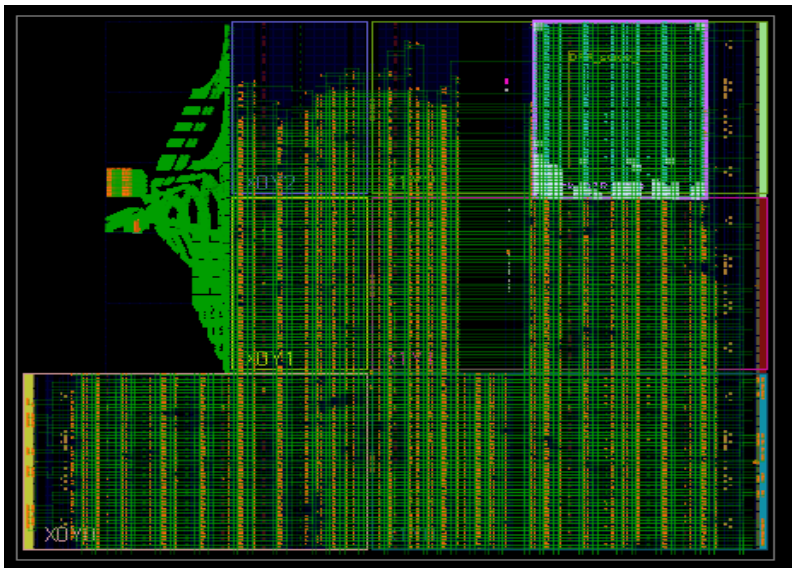


Figure 5.6 Implemented system floor-planning

To compare the resource utilisation with other implementation in the literature, the reconfigurable region resources count can be used to implement the entire ISP in a dynamic manner. In [150], the ISP utilises 41K logic elements of the cyclone III altera FPGA and 127 M9K. These values are approximately 9K Xilinx slices and 32 36K BRAMs. Our DPR implementation uses only 2K slices, 8K LUT, 40 DSPs, and 23 36KBRAM for the entire ISP in DPR manner. In [105], the Xilinx static ISP implementation utilises 31K LUTs, 52 36KBRAM, and 92 DSPs. Note that some of these IPP stages has been implemented in the processor side of the SoC. Table 5.2 shows these number in addition to other IPP implementations.

Table 5.2 The proposed DPR Architecure vs Other Architectures - Resource Utilisation

IPP stage	*LUT*	*RAMB18*	*DSP48*	*Slices*
Altera IPP Design [150]	(41K LE)	127 M9K = 32 36K	-	(41K LE), ~15K slices
Xilinx Design [105]	30,614	36 36K, 16 18K	92	-
ASICFPGA [104]	5302 / 6669	11 36K, 31 18K	-	11,170
Proposed Architecure	8,000	40	40	2,000

Based on Table 5.1, different sizes of reconfigurable regions provide different bitstream sizes. Therefore, the needed configuration time will vary from size to size. Table 5.3 shows the size of the partial bitstreams for each set of the RMs, and the configuration time for each bitstream employing the PCAP and using the configuration engine through the ICAP primitive. Bigger bitsreams need more time to be configured. The designed configuration engine will reduce this gap, which will increase the system performance, because the total time needed for configuring a stage will be reduced. The configuration engine is three times faster than the PCAP method, which is the default method in Zynq SoC.

Table 5.3 RMs Bitstream Sizes and Their Configuration Times

Design	Size of bitstream (KB)	Configuration time (ms)	
		PCAP	***Configuration Engine***
RMs Set 1 (1ppc)	436 KB	3.54	1.11
RMs Set 2 (2ppc)	581 KB	4.72	1.47

In terms of the total time, Figure 5.7 shows the timing for the system execution path. Obviously, it shows that the overhead in the system is the time for configuring a new stage. Table 5.4 shows the total execution time for the system and how the designed configuration engine reduces the overhead in the configuration. This overhead is proportional with the number of IPP stages. As shown in Table 5.4, a good improvement has been achieved when the stages are bigger and consume more logic resources.

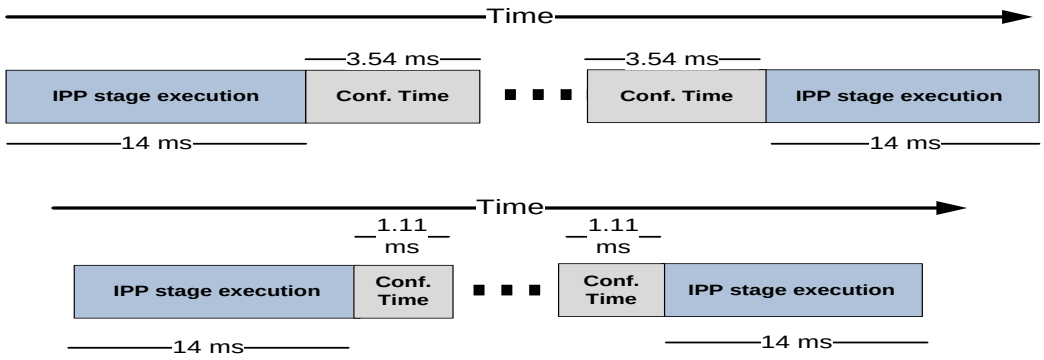


Figure 5.7 System total execution time and configuration time improvement

Table 5.4 Execution Time of the System for Number of IPP Stages

IPP stage	Time (ms)			
	Set 1 (1 ppc)		*Set 2 (2 ppc)*	
	PCAP	*Conf. Engine*	*PCAP*	*Conf. Engine*
CFA Interpolation + Median Filter	14.5	14.5	7.25	7.25
Configure next stage	3.54	1.11	4.72	1.47
Automatoc White Balance part1	14.5	14.5	7.25	7.25
Configure next stage	3.54	1.11	4.72	1.47
Automatoc White Balance part2	14.5	14.5	7.25	7.25
Total Time	50.58	45.72	31.19	24.69
Improvement	9.6%		20.8%	

As mentioned previously, the system is designed in a DPR manner, as only one of the stages exists on the FPGA logic at a time. This method allows an unlimited number of stages to be used dynamically. Conversely, the static design will allow the use of a few number of stages as the FPGA logic is not enough to fit all the stages at once. As a result, the total FPGA resource, which would have been used to design the system with DPR, is approximately 80% of the total resources. Therefore, it would not have been possible to implement the system as a static design with all the possible stages if the best quality algorithms are employed in each stage of the IPP. Table 5.5 shows the resource utilisation for the DPR implementation. Not all the resources are fully utilised except the LUTs.

To compare the system resource utilisation with the implementations mentioned in Table 5.2, note that the information in table 5.5 is for the entire system including IPP and other parts required for making the implemented system is so flexible to target different imaging application such as memory communication components, sensor interface, and display interface. IMPERX [107] has built their user customisable image processor in which an FPGA is used to build the image processing application. No information about the resource utilisation was provided in their manual.

Table 5.5 Resource Utilisation of the Implemented Design

Design	Resources / Utilisation			
	LUT	***Slices***	***RAMB***	***DSP***
DPR- (1 ppc)	39526 / 78%	10708 / 80.5%	43 / 30.7%	56 / 26%
DPR- (2 ppc)	41632 / 80%	11135 / 83.7%	48 / 34.3%	56/ 26%

- **Power Analysis:**

This section analyses the power consumption of the implemented DPR imaging system. It discusses the impact of different system activities on power over time. First, based on the outcomes of section 4.4.3, the employment of DPR in the design saves a significant amount of power compared to the static implementation. Based on this observation, the implemented DPR imaging system consumes much less power than the static implementation. The following methodology is used to evaluate the power dissipation on the DPR implementation. The power information is directly read in different system stages to see the effect of each stage on power. Examine the use of different techniques to reduce the power dissipation if possibel. TI Fusion Power Designer tool is used to monitor the power information on the Zynq board by communicating with the embedded power regulators and PMBus controller inside the Zynq board over the serial bus. The implementation is split into two parts between the PS and PL sides. As mentioned in section 4.4.3, the PS side consumes more power due to the generation of clock resources and the existence of the DDR memory interface. This work focuses on the PL side power consumption. Figure 6.8 shows the power consumption of the system over all possible system operations. The sudden power increases in the figure shows the power consumption when the design is processing data, while the lower consumption levels shows the consumption at the idle state. Due to the low polling rate in the tool, the readings cannot be obtained in the normal case. Each module was executed 600 times to obtain these readings. The figure is divided into five sections based on the system state. The part with label A shows the power consumption when the imaging system is ready to capture the image where the image appears on the user screen after passing through the simple

IPP on the design. In this state, the data has to pass through the buffer on the DDR memory. The parts with labels B, C, and D show the power consumption when the system imaging tasks are processed. The difference in power consumption is due to the nature and the activities of these tasks. For example, the tasks with labels C and D are primarily executing division and multiplication operations via DSP components while the task with label B is using LUT and BRAMs components for its operations. The amount of region utilisation is also affecting the amount of power dissipation. The part with the E label shows the power consumption when the system at idle state.

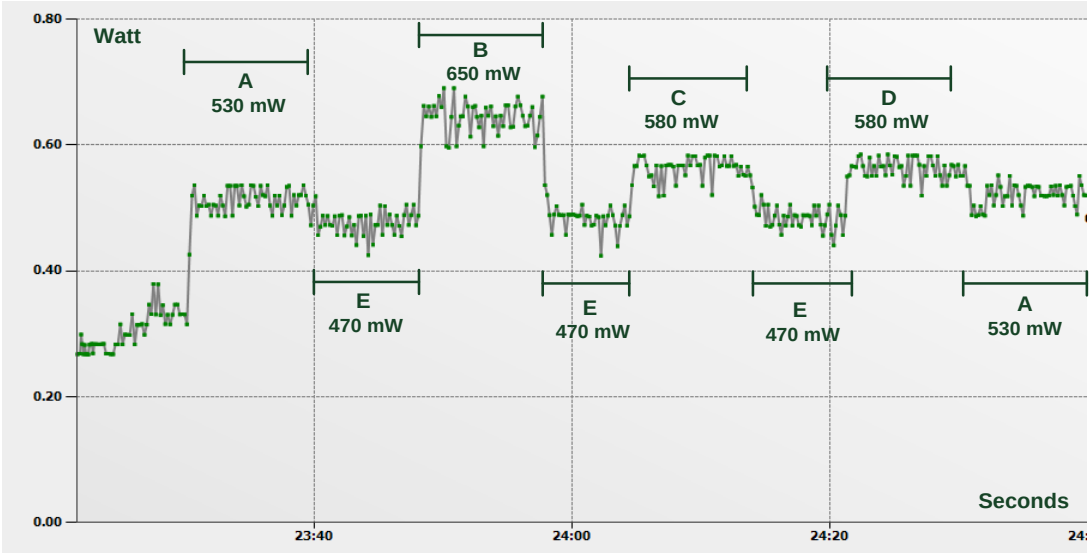


Figure 5.8 Power consumption of the implemented DPR imaging system over different states

- **Power Optimisation:**

In Figure 5.8, the part with label A has extra 60 mW power consumption compared to the part of the figure with label E. This extra consumption is because the data are directed constantly to the DDR buffer through DMA. The constant use of DMA consumes extra power. To reduce the power consumption and keep it at the level of idle state, the design has been re-implemented to use the DDR memory buffer only at the time of capturing an image. When the capture button is pressed, the image data are redirected to the DDR memory for further processing by the system. Figure 5.9

shows the power consumption in the new design. It is clear that label A and E have the same consumption level. It is important to save this amount of power because the system spends most of the time in this state. The power readings in this figure represent the total power dissipation of the customised imaging system over different states including the static and the dynamic power. Other similar implementations did not include the power consumption analysis except the work proposed in [150]. It consumed approximately 1258 mW by the IPP and the surrounding components while our implementation is consuming 550 mW in average.

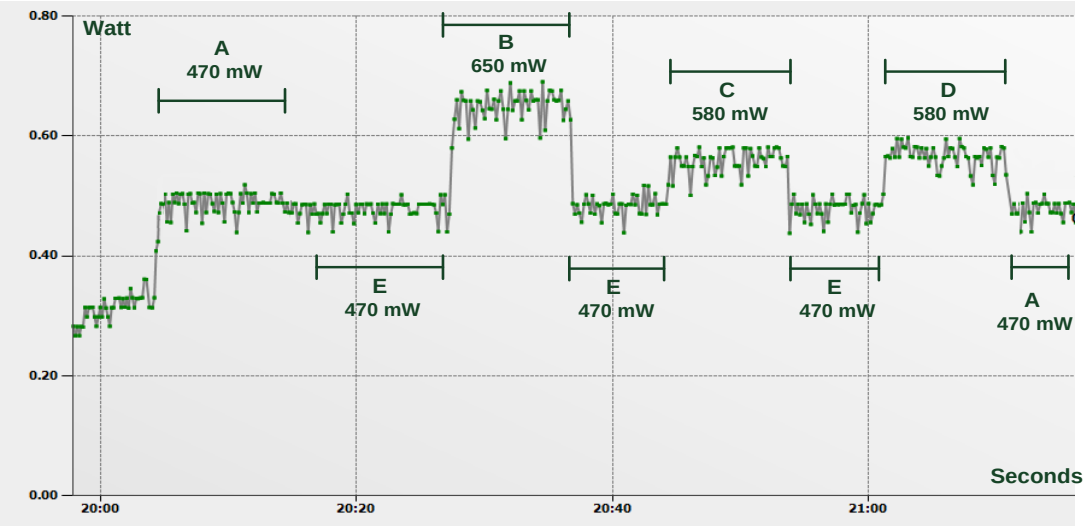


Figure 5.9 Power Optimisation for the implemented DPR imaging system over different states

5.3 System Enhancement: Task Pre-fetching

Referring to Table 5.4, the process of configuring new tasks can add extra time overhead to the system, which leads to a reduction of the system overall performance. As shown in that table, the configuration time can be 30% of the total system processing time. In this section, a task pre-fetching is used to mask the configuration time and to eliminate any chance of system performance reduction because of that. The idea is to add an additional reconfigurable region on the FPGA that works the same as the existing reconfigurable region. The two regions should work in a way that masks the configuration time. The system will divide the

activities into two sets: foreground activities and background activities. In this context, the configuration time is masked by configuring the next task in the background on the empty reconfigurable region while the system is executing the other task in the foreground. When the foreground task is completed, the system will immediately activate the newly configured task by swapping the control over the two regions and configure the next task in the background. Figure 5.10 shows the enhanced version of the DPR imaging system with two reconfigurable regions and swapping mechanism.

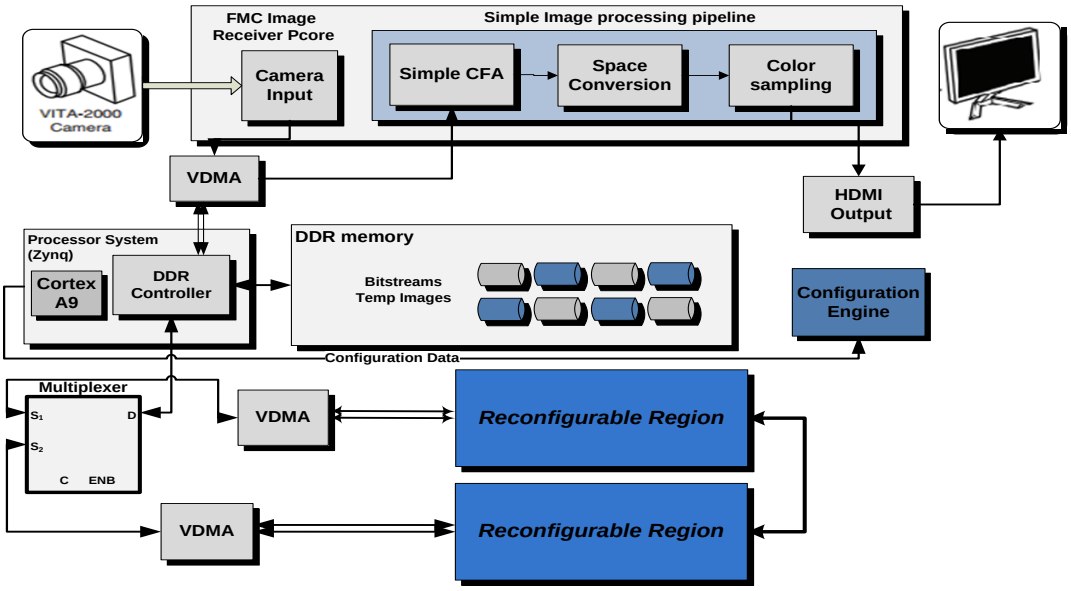


Figure 5.10 Enhanced DPR imaging system with two reconfigurable regions

In addition to masking the configuration time, the enhanced version of the system can be used to enhance the following aspects:

- **Parallel execution**:

Because of the existence of two reconfigurable regions, the system can place two tasks at the same time on the FPGA. These tasks can be executed in parallel to increase the overall system performance. As shown in Figure 5.10, the system has two VDMA cores to enable this feature; each reconfigurable region has its own path

to the memory to fetch and write the data easily with the maximum speed. Based on the available technology, the system can reach a bandwidth of 4800 MB/s for the reconfigurable regions because they are connected to the High Performance (HP) ports available on the Zynq-7000 boards. Each HP port is able to read or write data from the DDR memory with a bandwidth of 1200 MB/s based on section 22.3.1 in [11].

The executed tasks can be dependent or independent tasks. As shown in the aforementioned figure, the two tasks are linked using a high-speed direct connection that makes it possible to share intermediate data while processing the tasks. In the case of independent tasks, the task can be executed immediately after configuring the bitstream on the FPGA; it needs to configure the other task in the case of dependent tasks. Figure 5.11 shows scenarios where dependent and independent imaging tasks are used in the system. The first three tasks in each scenario depend on the output data of the previous task, while the fourth task in each scenario is either task-independent or dependent.

- **Variable task size**:

In the case of tasks that need a bigger region to hold their logic, the previous proposed implementation cannot handle this type of tasks. A good example for this situation is the JPEG core, which needs a minimum of 9,000 LUTs on the region compared to the maximum amount of LUT that the system can provide (which is 8000 including the 20% of additional resources based on table 5.1). The only solution of the previous case is to increase the total resources in a single reconfigurable region to be able to hold the biggest core. This is not a good practice in the case of having many small tasks and only one or two big tasks, as the remaining logic in the reconfigurable logic will be wasted when managing the small tasks. In the enhanced version of the system, the relatively big tasks can be split into two tasks working in parallel, based on the previous feature with the ability to write the data to one of the tasks and read the results from the other task, as shown in Figure 5.12. In this figure, the VDMAs are connected to the memory interface through a multiplexer to choose between the two paths to enable this feature. Once

the path is enabled, the system will ignore the two connections: the read port in the first reconfigurable region and the write port in the second reconfigurable region. This practice will save much space in the FPGA and simultaneously enables the system to configure multiple task sizes. Moreover, in the case of big task deployments, the two parts of the task must be fully configured (see Figure 5.11.b) before the system triggers the controller to start the execution process, otherwise the system will stop working at some stage. In the context, the tasks have to be configured sequentially; they cannot be configured at the same time due to the use of only one configuration port.

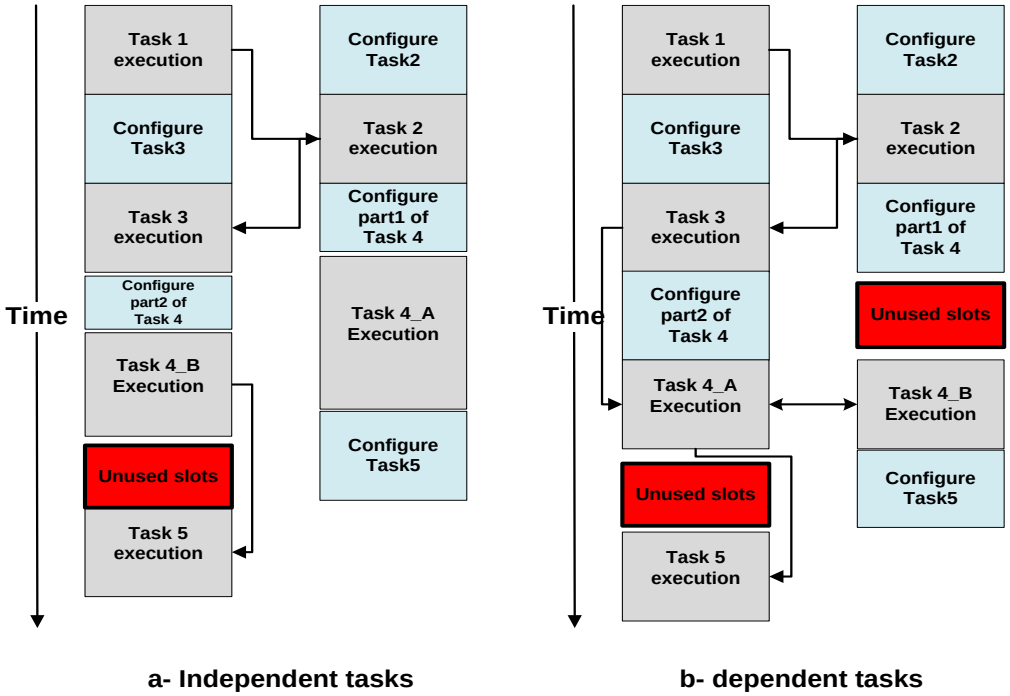


Figure 5.11 Independent and dependent tasks in dual region proposed imaging system

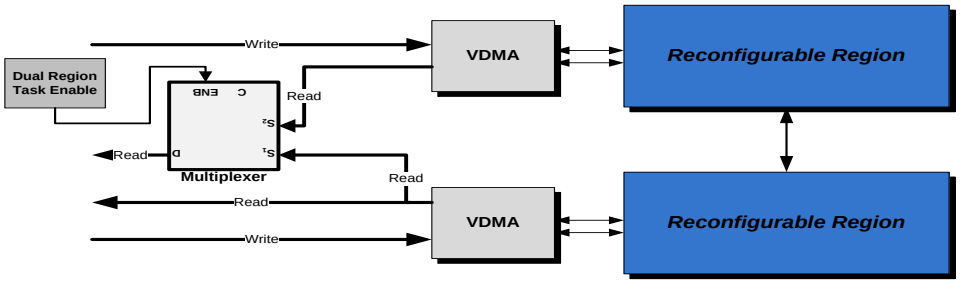


Figure 5.12 Dual reconfigurable region path multiplexing

5.3.1 Implementation

The enhanced version of the system has a similar structure to the single reconfigurable region implementation except regarding the additional reconfigurable region. Similar tools and similar implementation steps have been used to implement the proposed system. The DPR flow should be applied to the two RRs. Each task should have two different bitstreams, one for each reconfigurable region, to increase the flexibility of the system by enabling the task to be allocated to any of the regions based on the application requirements and the allocation algorithm.

In the proposed system, the two reconfigurable regions have to be isolated at the time of configuring new tasks with relatively similar steps to the isolation process of the single RR implementation. Because of the existance of additional regions, the isolation and configuration processes should follow the pseudo code presented in Algorithm 5.1 and the data flow chart shown in Figure 5.13.

```
                                        //Task 1 in region 1
VDMA1 start - Write data
Execute task
VDMA1 start- read data                  // continuous read and write
        Isolate region 2
        Configure tasks 2 in region 2   // Through different memory path
        Release isolation signals
Keep polling until done signal
Stop VDMA1 write- read

                                        // Task 2 in region 2
VDMA2 start - Write data
Execute task
VDMA2 start- read data
        Isolate region 1
        Configure tasks 3 in region 1   // Through different memory path
        Release isolation signals
Keep polling until done signal
Stop VDMA2 write- read
.....
Keep repeating until the last task
```

Algorithm 5.1 The isolation and configuration process of Dual-reconfigurable region

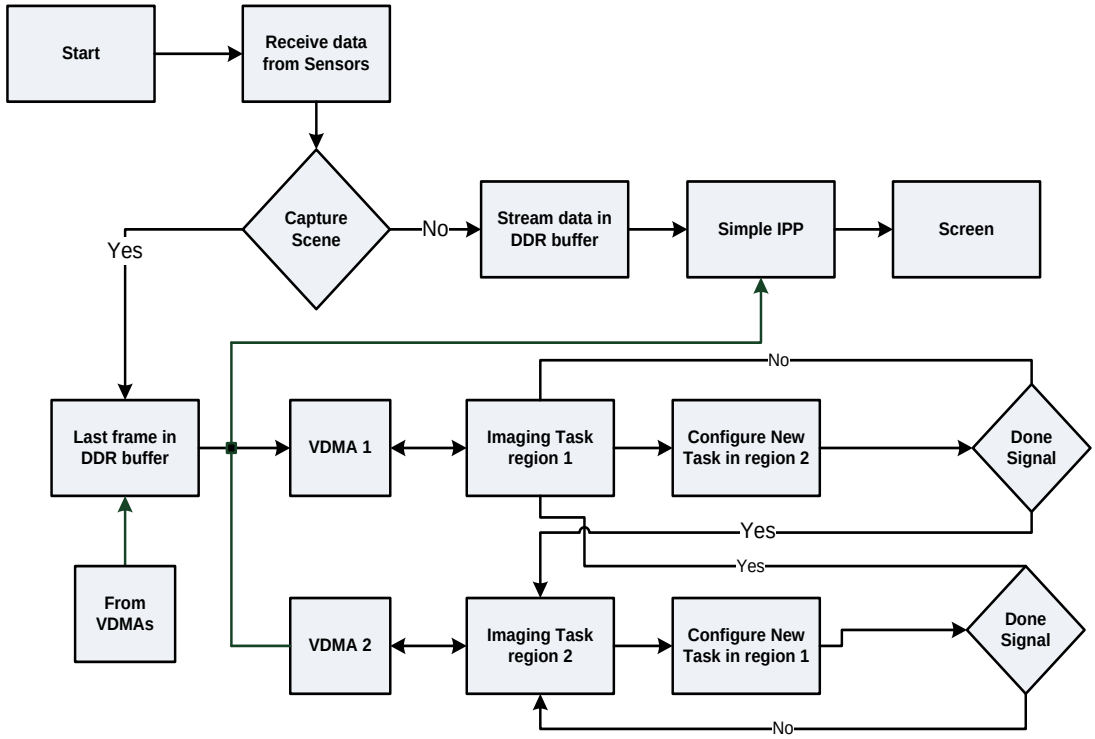


Figure 5.13 Data flow of the enhanced version of the DPR imaging system

5.3.2 Experimental Results

Similar to the single reconfigurable region implementation, the enhanced version of the system has been tested with three different tasks. The resource used by each reconfigurable region and the size of the reconfigurable region are similar to the information provided in Table 5.1. Moreover, the information provided in Table 5.3 can be applied again to this version of the system by using the implemented configuration engine.

Unlike the single reconfigurable region design, this version of the system enhanced the overall performance by masking the configuration time shown in Table 5.4. Table 5.6 shows the new execution time after using the new version of the system. Note that the use of any configuration method to configure the next task does not affect the system performance because the configuration time is encapsulated within the time of executing the tasks due to the use of pre-fetching method. This method employs two reconfigurable regions that are revealed in Figure 5.14, which shows

the floor planning of the enhanced version of the system. Note that the two regions are connected using a direct link to share the data between the two tasks online as shown in the figure. The execution time of the system depends on the amount of system parallelism.

Table 5.6 Execution Time of the System for Number of IPP Stages

IPP stage	Time (ms)	
	Set 1 (1 ppc)	***Set 2 (2 ppc)***
CFA Interpolation + Median Filter	14.5	7.25
AWB part 1	14.5	7.25
AWB part 2	14.5	7.25
Total Time	43.5	21.75

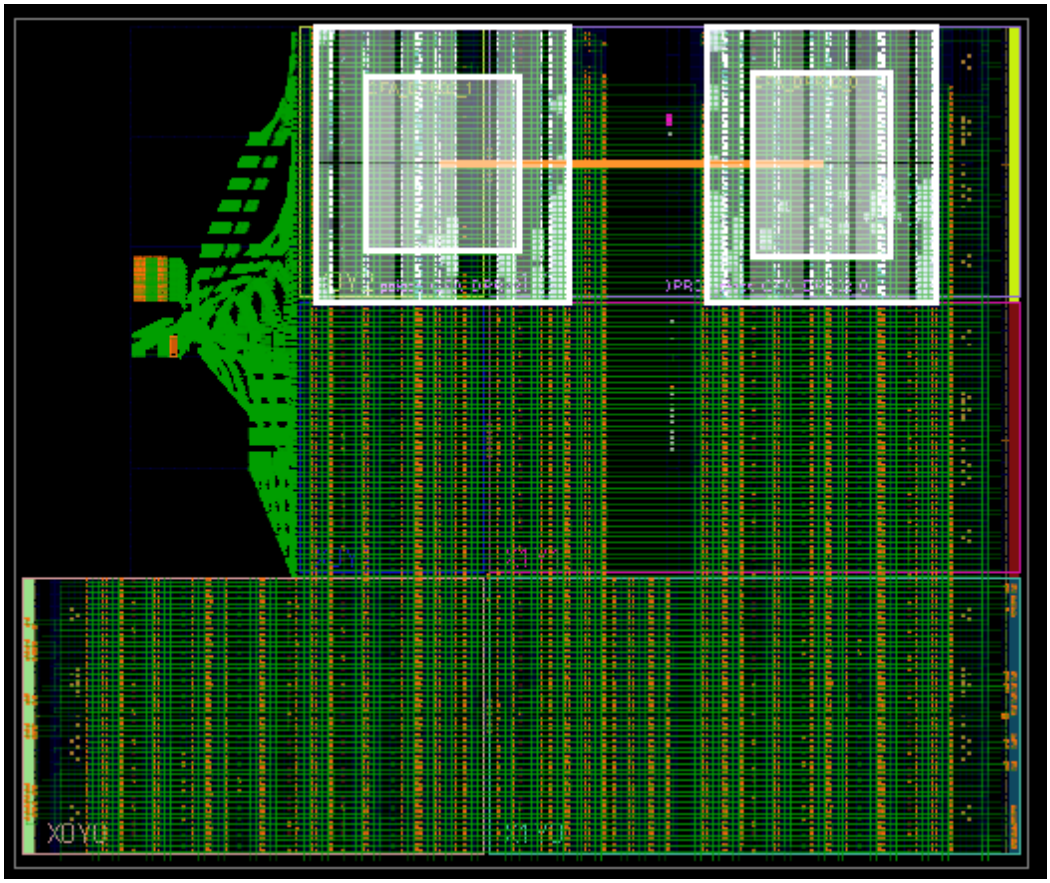


Figure 5.14 The enhanced system floor-planning

Similar to Table 5.5, Table 5.7 shows the total resource utilisation of the enhanced system. In this implementation, the design utilised more than approximately 90% of the slices and LUTs. As mentioned previously in this chapter, this practice allows the use of an unlimited number of imaging tasks by swapping in and out in the reconfigurable regions, while it can fit only two tasks by using the static implementation. In addition, the information provided in this table are still in the range of the previous implementation with an enhanced performance.

Table 5.7 Resource Utilisation of the Implemented Design

Design	Resource/ Utilisation %			
	LUT	***Slices***	***RAMB***	***DSP***
DPR- (1 ppc)	47685 / 89.6%	12308 / 92.5%	65 / 46.4%	96 / 43.6%
DPR- (2 ppc)	51575 / 96.9%	12811 / 96.3%	71 / 50.7%	96/ 43.6%

The power consumption of the dual region implementation has been measured under different activities, including task execution and online bitstreams configuration. The tests showed that the enhanced system has similar power results as the old version with minor differences. This shows that enhancing the performance of the system does not affect the power consumption too much. The design has been modified to reduce the power consumption of using the DDR memory buffer constantly through VDMAs to keep intermediate data in the memory. The modification shows that redirecting the data to the buffer only at the time of capturing the image saves significant power, as shown in Figures 5.15 and 5.16. Compared to Figure 5.7, the difference in the amount of power consumption in part A is because of the use of two VDMAs in the enhanced version of the system compared to one VDMA in the single region implementation.

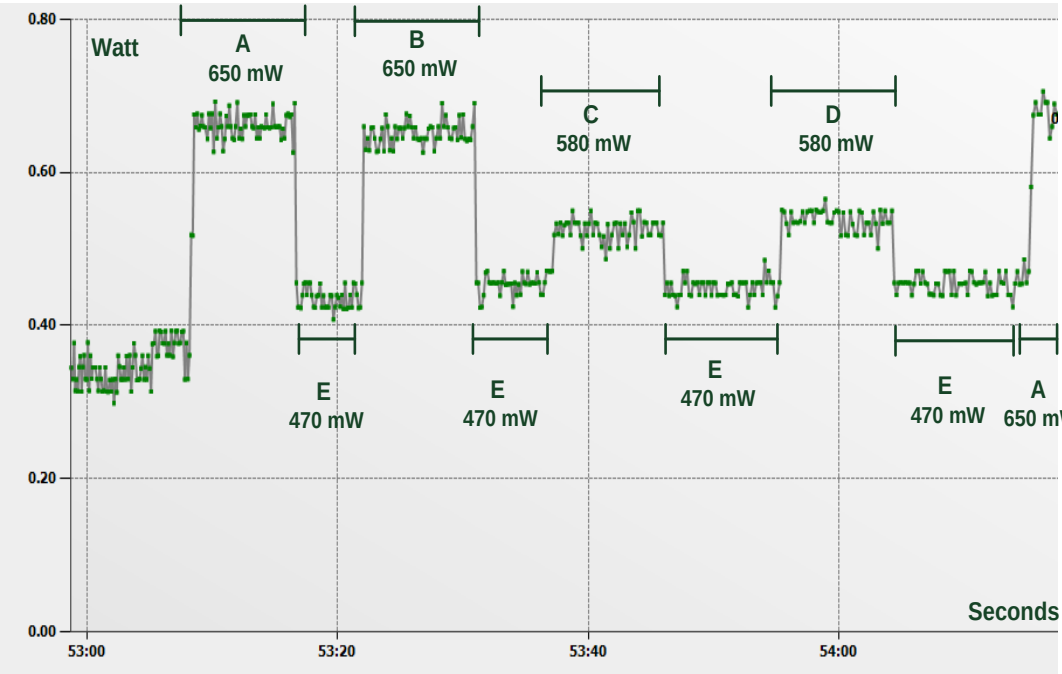


Figure 5.15 Power consumption of the enhanced system before applying modifications

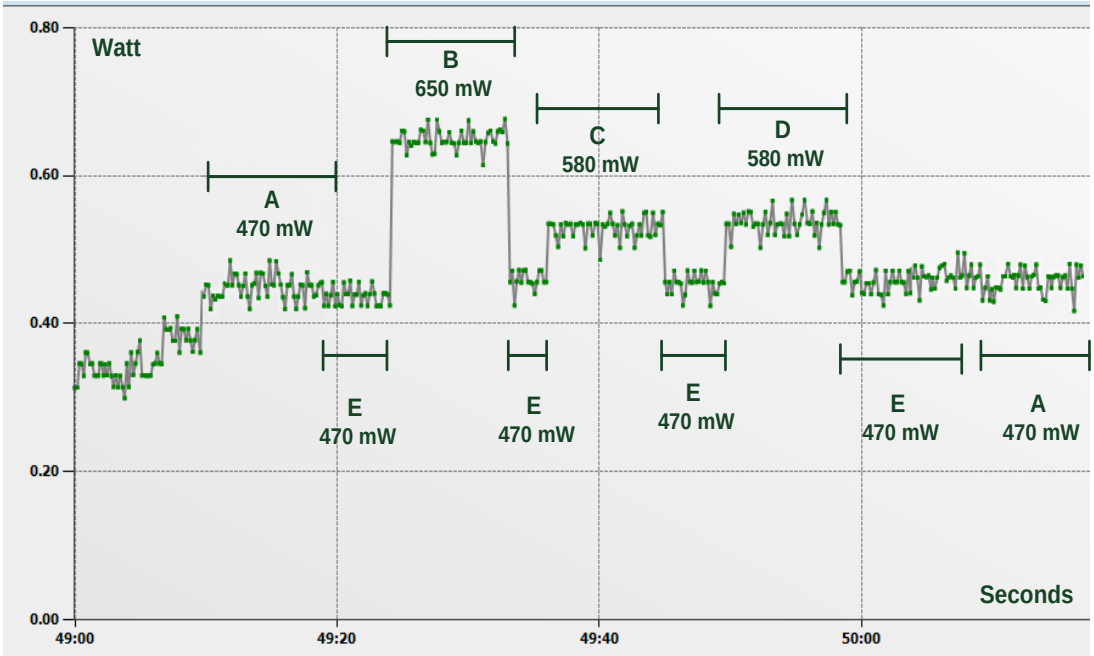


Figure 5.16 Power consumption of the enhanced after applying modifications

5.3.3 System Comparison and Evaluation

This section discusses the advantages of this implementation over other available imaging systems. Because of the lack of technical information of the current available imaging systems, the section shows the added features of the system over others. If the technical information is available, a comparison is made to find out these advantages.

Most current commercial modern cameras are built on ASIC platforms, where all parts of the system have to be placed together on the platform to form a static-based system in which all parts cannot be changed or replaced in the future after fabrication. An example of such architectures is presented in all modern camera processors such as EXPEED, DIGIC, and EXILIM for Nikon, Canon, and Sony respectively. The component corresponded to each task is placed in specific place on the chip die. The ASIC-based implementations have excellent performance, power consumption, and area utilisation but they are suffering from lack of flexibility, which prevents them from updating or adopting new standards or technologies. The same issue is partially presented in [150], [105], and [104]. The IPP core and other parts are implemented as static design, which prevents the change of any part of the system unless the entire system is reconfigured by uploading a new bitstream. This will stop the operation of the entire system until the new configuration is uploaded.

In our implementation, the system is flexible allowing the user to change or replace tasks with new versions so it can protect the product from obsolescence. However, the static-based imaging design has to include all possible image processing tasks on the design logic to be able to use them later while our implementation has to include only the minimum parts to execute one task because the tasks can be swapped in and out using the DPR feature, which minimises the area utilisation. In this context, utilising less area leads to less overall power consumption, as discussed in section 3.3.2. Moreover, the flexibility of this system has no limit because this platform can be customised to build similar imaging application. This process cannot be accomplished in most imaging systems with the same efficiency. The application can be customised by developing individual tasks and keeping their configuration files in

the DDR memory. Based on the application requirements, these bitstreams can be fetched and configured on demand. To compare the system flexibility, the IPPro soft-processor presented in [109] has been proposed to process imaging tasks in which a number of soft-processors execute instruction using the load-store method. This type of platform has a good flexibility but needs more effort in building a programming environment to compile the instructions to be used by the soft processor. The dependency between the instructions can limit the performance of such platforms. Moreover, as shown in the work, the area utilisation required for selected example was too much compared to the expected area utilisation for equivalent example using our method. Other platforms either have lack of flexibility or cannot provide a high-performance environment for imaging systems.

The idea presented in [152, 153] has similarity to our idea in general but there are some differences. The architecture was implemented on the first Xilinx FPGA that supports DPR, Virtex-II Pro XUP board with embedded Power PC processor. Generally, the implementation experimental results reflect the FPGA poor capabilities compared to the current FPGAs. The system is clocked at 100 MHz and can configure bitstreams using a clock of 50 MHz. These two values will reduce the overall system performance. The system uses two reconfigurable areas in addition to a reconfigurable on-chip (ReCoBus) and I/O bar primitives to provide a communication infrastructure for the dynamically loaded modules. Tracking human motion case study for the system has been demonstrated to analyse the system flexibility. The system most likely assumes the FPGA has a homogeneous architecture to place the module anywhere in the reconfigurable region. The problem here appears when the system needs to adopt different application. To change the application, the module must be configured in any place in the reconfiguration area. This needs a relocation feature within the system. Unfortunately, modern FPGAs are heterogeneous FPGAs that have different elements distributed over the FPGA plane. For the implemented system, it is simple to relocate small modules within the reconfigurable region due to the small number of resource columns, but big modules has more number of columns that need a specific resource layout in term of CLB, BRAM, and DSP primitives. It is hard to find this layout unless the modules have

fixed places within the reconfigurable area. This means less overall system flexibility. Moreover, due to the use buses to connect the modules, most of the time is spent on connecting the interrupts and bus requests that set the alignment parameters. Furthermore, the used NPI memory interface has additional time overhead as discussed in chapter 3. In [115], a coarse grained architecture divides the pipeline into three fixed sub-modules configured sequentially on top of pre-defined elements. The ASIC like architecture enables the system to reduce the power dissipation but the system has a time overhead to configure the new sub-modules. Moreover, the sub-module has a fixed functionality, which limits the system flexibility to a few numbers of applications. Table 5.8 shows a comparison between our architecture and other imaging systems based on different aspects.

Table 5.8 System Comparison with similar appraches in the literature

Architecture	Performance	Flexible	Power dissipation	Application Customisation	Area utilisation	Style
Modern Camera Processors	High	Not flexible	Low	No	Low	Static
Altera [150], Xilins [105], ASICFPGA, [104],	High	Partially	Medium	No	Medium	Static
DPR Camera [152, 153]	Average	High	Medium	Limited	Medium	DPR
CRISP [115]	Average	Partially (fixed modules)	Low	Limited	Low	Reconfigurable
IMPERX user Processor [107]	Average	High	Unknown	Limited	Low	Programmable
IPPro Soft processor [109]	Average	High	Medium	Yes	Medium - High	Soft processor
Our Architecture	High with parallelism	High	Low	Yes	Low	DPR

5.3.4 Output Images

Figures 5.17, 5.18, and 5.19 show the images after different stages using the implemented imaging system. The quality of the images is not discussed in this

section as the system can change the algorithms used in each task dynamically based on the requirements of the application, which is the purpose of the system. More tasks can be used to get different images or to get specific information from the images data, which can be used later in other aspects, as shown in a later section in this chapter.

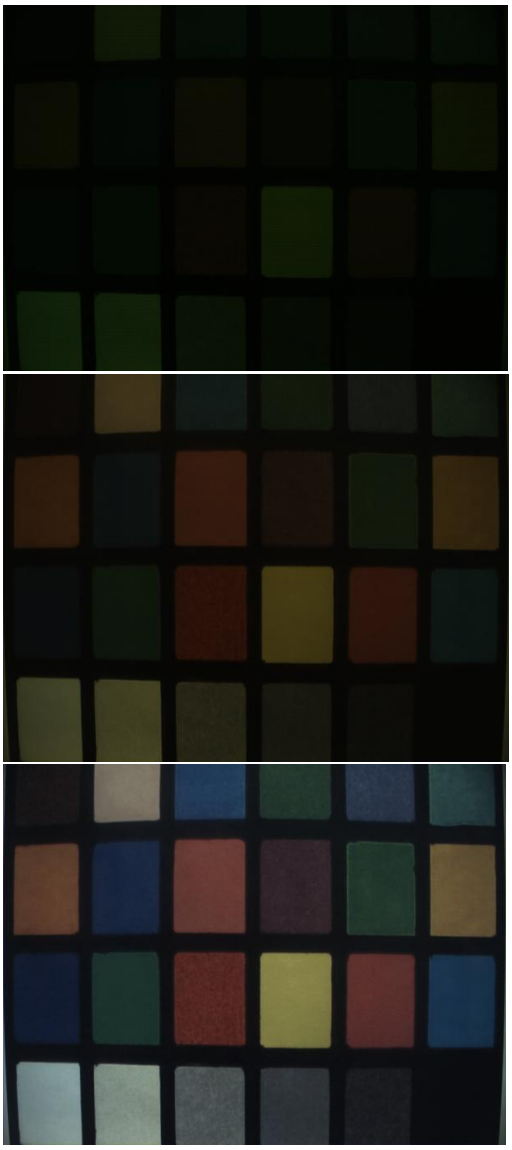
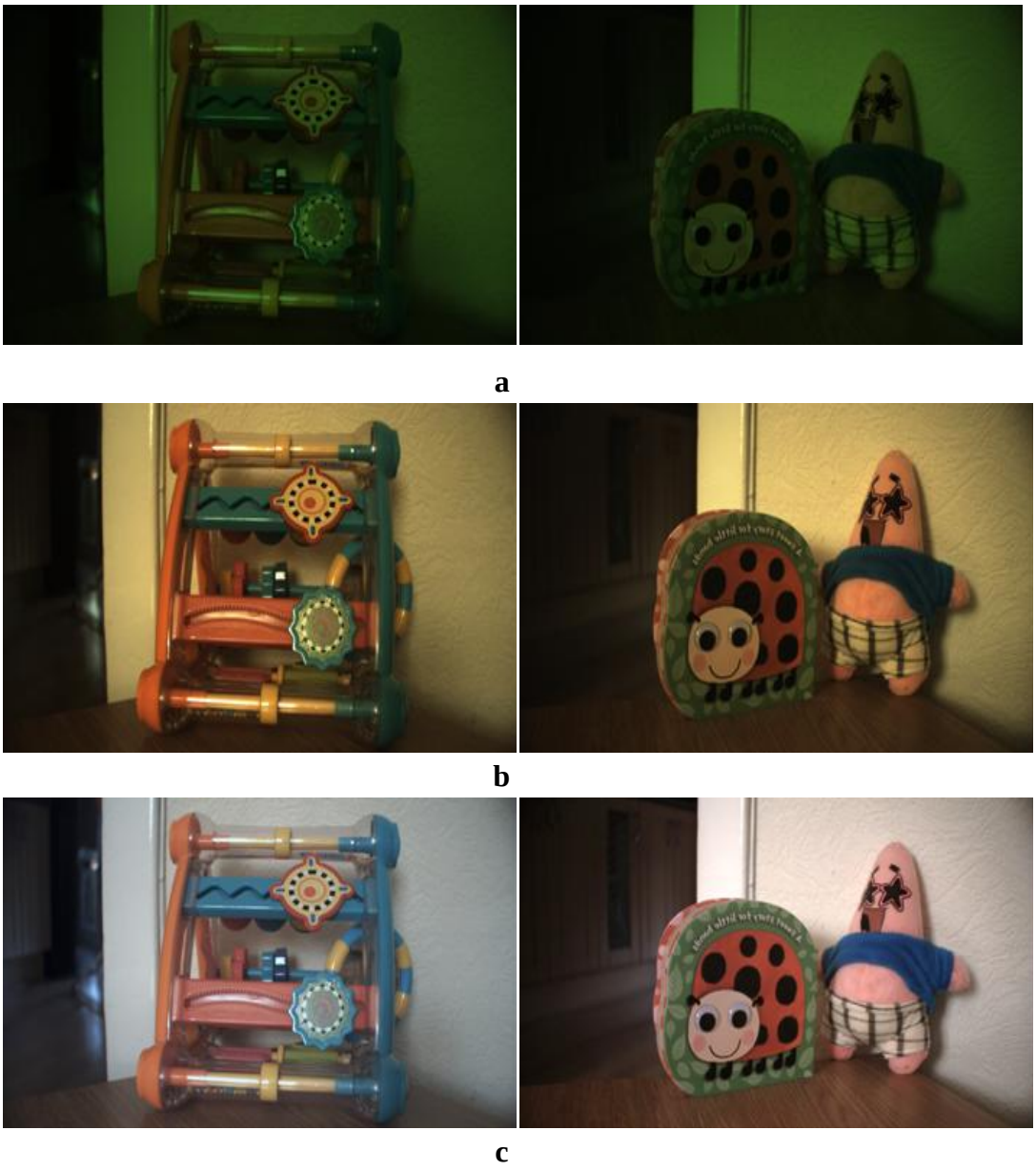


Figure 5.17 Output images. (a) raw image (b) Colour filter array interpolation output (c) AWB output

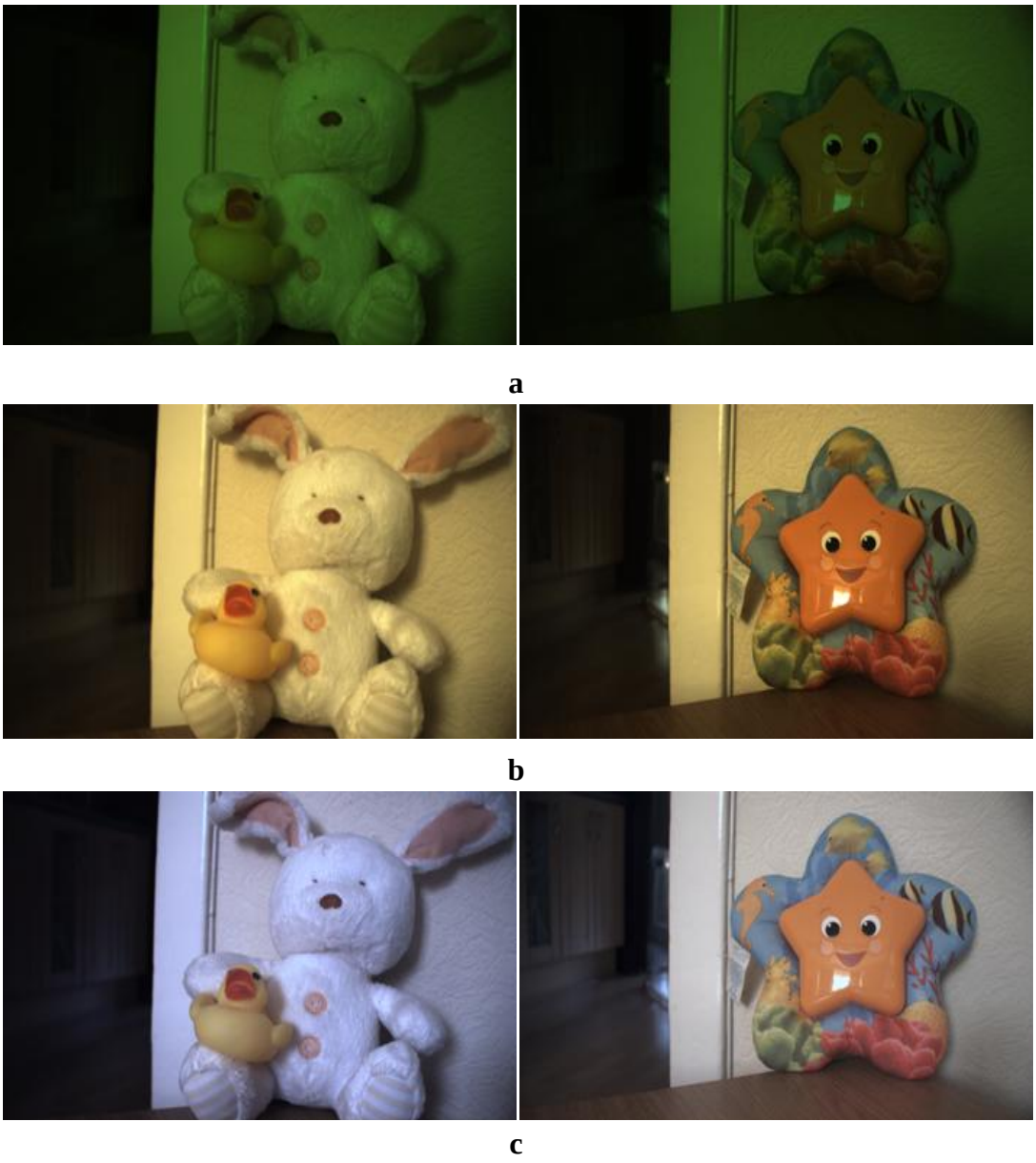


a

b

c

Figure 5.18 Output images. (a) raw image (b) Colour filter array interpolation output (c) AWB output



a

b

c

Figure 5.19 Output images. (a) raw image (b) Colour filter array interpolation output (c) AWB output

5.4 System Integration with R4THOS for Autonomous Cars

Before explaining the system integration in detail, the work proposed in [157] shows a relatively similar reconfigurable architecture that has reconfigurable region to accommodate the image processing hardware accelerator engines. These engines work as video-based driver assistance system by swapping the modules in and out based on environmental conditions. The proposed system is so small that has only few modules to demonstrate the idea, but with the increasing of system complexity, controlling these modules will be hard and needs much effort from a higher system level. To use the implemented imaging system in an efficient way, the system has to be integrated into a complete environment that manages the system activities alongside the other surrounding components. The activities of the system are mainly the actions of swapping tasks in and out based on the application requirements. These activities should be managed from a higher level of abstraction that looks to the implemented system from the top and knows the activities of other components so it can manage all activities based on the relations and dependencies between the parts making up the system. In this context, this section discusses the integration of the imaging system alongside other sensors with the proposed R4THOS to form a central core that controls all activities of autonomous cars. Figure 5.20 shows the parts of the proposed system. As shown in the figure, the cameras, alongside other sensors, are placed on the outside surfaces of the car to sense and to gather the information needed to help the car navigate itself. The proposed system divides the FPGA into two parts: the first part is the static part in which the R4THOS is placed, and the second part is the dynamically reconfigurable region for hardware tasks. The activities of swapping tasks in and out are controlled by the R4THOS component. The idea is similar to the idea of the operating system in PCs, where tasks are processed based on specialised algorithms in the OS. The Zynq-based R4THOS is divided into two main parts: hardware microkernel (HW μK) and software microkernel (SW μK). The SW μK is running on the PS side of the Zynq-7000 board where a dual-core ARM Cortex-A9 is placed. The HW μK is running on the PL side of the Zynq-7000 board. The functions are distributed between the two kernels based

on the amount of the computation of each function. As shown in Figure 5.21, the HW µK of R4THOS consists of three main components: allocator, configuration manager, and security manager. The SW µK of R4THOS consists mainly of the scheduler, which manages the scheduling of hardware and software tasks. A hardware-software interface is placed between the two kernels to connect them using the high-speed ports available on the Zynq-7000 board to connect the PS and PL sides.

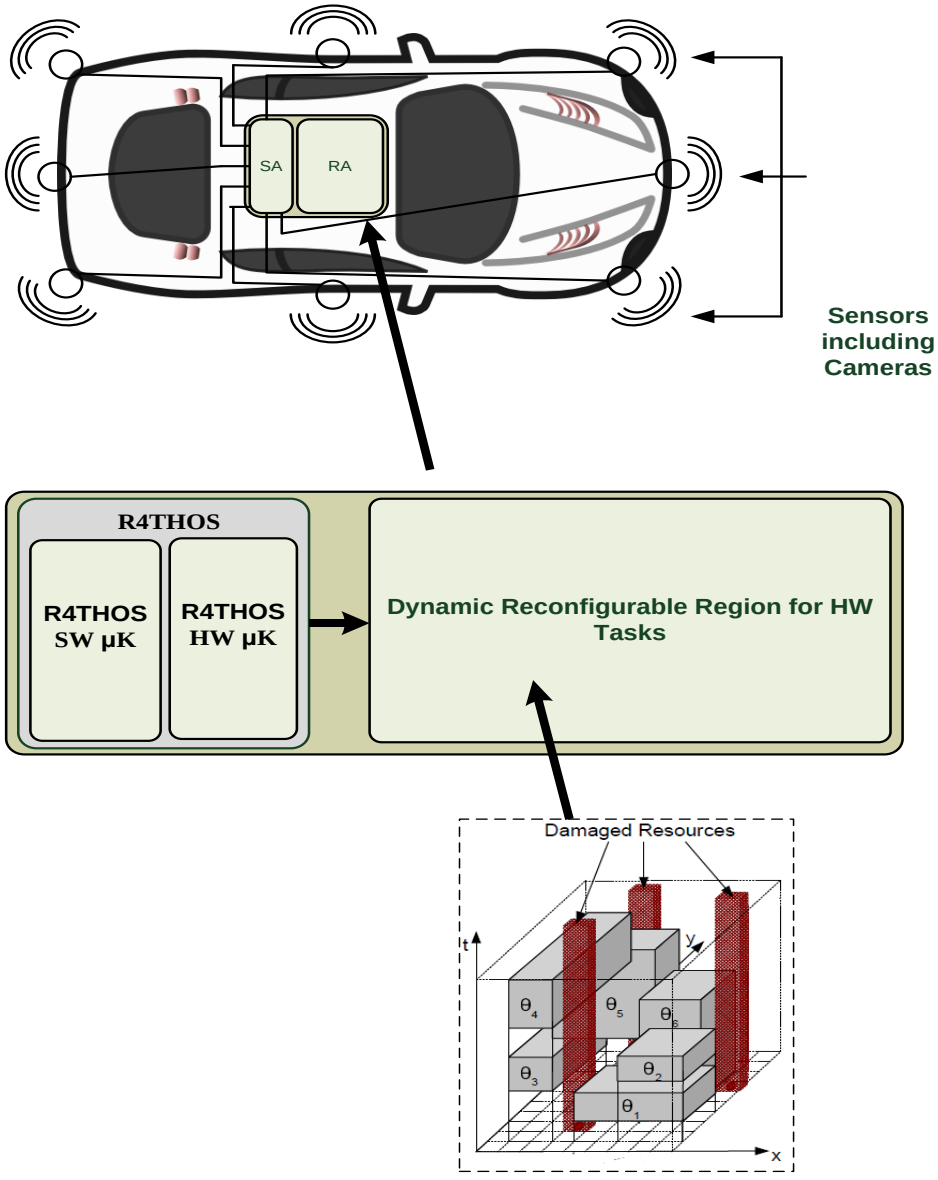


Figure 5.20 R4THOS for autonomous cars

5.4.1 R4THOS

R4THOS is a hardware-based reconfigurable operating system, which exploits the FPGA reconfigurability to create an infrastructure for executing hardware tasks dynamically, reliably and in real-time. The reconfigurable operating system aims to hide the complexity of reconfigurable device to ease the development of FPGA-based high performance reliable applications. In R4THOS, the mean of the reconfigurable operating system is proposed by swapping in and out the hardware tasks using the DPR feature supported in modern FPGAs. Moreover, the use of a reconfigurable OS eases the management of hardware tasks and provides a software look and feel to the user. R4THOS is the enhanced version of R3TOS [158] where security features have been added.

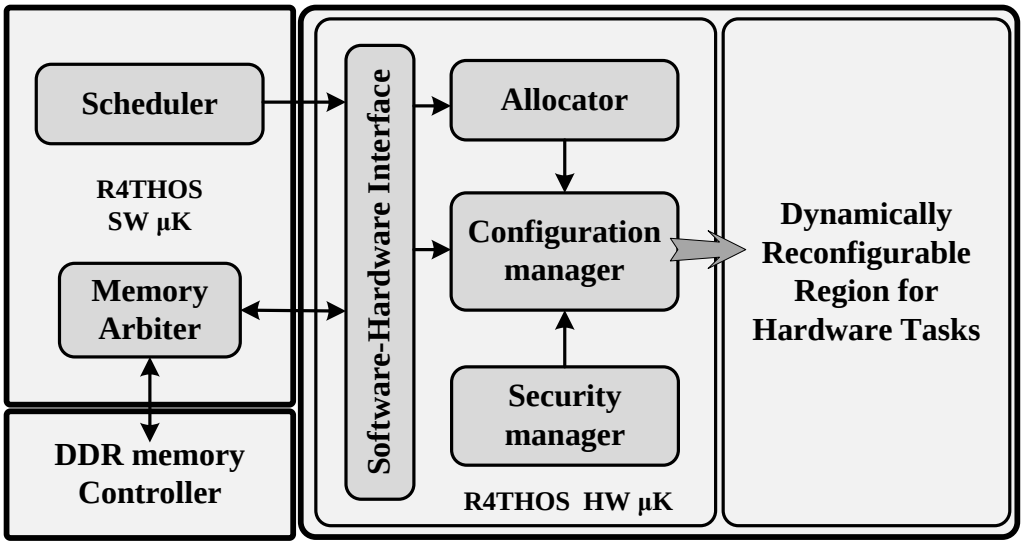


Figure 5.21 The R4THOS architecture showing major functional modules

R4THOS, as mentioned previously, consists mainly of the scheduler, allocator, configuration manager, and security manager in addition to the ARM processor that resides in the PS side of the Zynq board, which can process the scheduled software tasks. R4THOS is still under development by System Level Integration Group (SLIG). The following paragraphs discuss each part and its functions.

Scheduler: Although it can be placed within the HW μK part of R4THOS, the scheduler is placed in the SW μK for designing purposes. The scheduler is expressly designed to manage the scheduling of hardware and software tasks. To fulfil the timing requirements in the reconfigurable OS, a real-time scheduler is used. The scheduler coordinates access to the configuration memory of the FPGA in an efficient way by assigning a suitable allocation starting time to the queued tasks and the order of tasks. The scheduler uses specialised scheduling algorithms that employ a number of factors such as task priority, task size and available area, finishing time of current tasks, and others. Many algorithms have been used in different stages of developing R3TOS and R4THOS. Implementation of Earliest Deadline First (EDF) [159] is proposed in [160], where scheduling is based on the task deadline. An enhanced version of the algorithm, called Finishing Aware EDF (FAEDF), is proposed in [161]; it can look ahead to find future releases of adjacent pieces of area when executing tasks finish.

Allocator: The allocator is designed to find and assign a suitable location on the available resources on the dynamically reconfigurable region for task execution. For performance reasons, the allocator is placed in the HW μK part of R4THOS. Generally, to get the most out of the FPGA reconfigurable region, the allocation algorithms should reduce the FPGA plane fragmentation. This should be achieved by placing the hardware tasks close to each other to expand the available space on the reconfigurable region in a given time. The allocation algorithms are developed based on the 2-D nature of the hardware tasks. The following algorithms have been used in different stages of the proposed Reconfigurable OS: Empty Area Compaction (EAC) and Empty Volume Compaction [160, 162] and [161]. Another allocation strategy (called snake) [163] is developed to enhance the performance by reusing the previously configured circuitry and the intermediate partial results between computation stages.

Configuration manager: The manager is the core component of R4THOS; it links the R4THOS parts with the FPGA configuration memory. In other words, it translates the high-level operations ordered by the allocator and scheduler into

reconfiguration commands for the FPGA through the ICAP primitive. These commands include task reconfiguration, task relocation, error detection and correction, and inter-task communication. For more details, the configuration manager has several configuration operations:

- ✓ **Partial bitstream configuration**: This is a general bitstream reconfiguration for RMs in a single RR. It is similar to the operation done by the configuration engine proposed earlier in this chapter and in chapter 5.
- ✓ **Partial bitstream relocation**: This operation relocates modules to any place in the FPGA online by modifying the location's binary data within the bitstream to be placed into a different location. This helps the ROS to place the hardware tasks anywhere in the plane of the FPGA based on the allocation algorithm.
- ✓ **Black-box configuration**: Basically, this operation removes all the logic configured in the region for that specific module apart from the static routes that pass in the region. This operation occasionally helps ROS to remove the logic after that task is terminated for power reduction.
- ✓ **Multiple-clone configuration**: This operation speeds up the configuration process of configuring multiple identical modules on RRs that have the same resource column order. This operation uses the MFW feature supported in Xilinx FPGAs.
- ✓ **Frames read and write**: This operation reads and writes individual frames from different locations within the FPGA. This operation is used in error detection and correction and is supported by the configuration manager. The idea is to read a frame each time, and examine the embedded parity bit in that frame for any errors and write back the frame with any possible correction. This operation is called scrubbing; it reduces the soft errors, which are discussed in the next chapter. This configuration manager can replace the SEM IP core discussed in chapter 7 for soft errors detection and correction.

The configuration manager for R4THOS is being developed currently by SLIG. The configuration manager belongs to R3TOS is proposed in [46, 69, 164].

5.4.2 System Functionalities

Although this section should present the proposed system functionalities and experimental results, the system has not been completely demonstrated because of the absence of a complete version of R4THOS. This section presents possible system functions and different scenarios for such application. The proposed reconfigurable region of the DPR imaging system should be placed in the reconfigurable region of the R4THOS to be able to reconfigure the tasks dynamically. Moreover, the static part of the imaging system has to be placed in the reconfigurable region also, as R4THOS is dynamically interfacing the system with the suitable ports and input clocks.

To enable the car to navigate itself on the road, the system core has to be prepared with all possible tasks that automate the car navigation. Each action of automation can be achieved using one, two, or several specific tasks. Moreover, besides the camera sensors, the car has to be equipped with other sensors, which can work together to automate the car navigation by getting different type of information. These sensors can be summarised as follows: radar to track the other vehicles, compass, aerial to obtain the location using GPS, ultrasonic sensors to detect the movements of objects. The camera sensors can be used in different ways to obtain different readings. It can be used to detect and identify any objects around the car, recognise and read the traffic signs, detect traffic lights, measure the distance with objects in front and the rear of the car, and others. Any functionality can be achieved through information gathered from one sensor or more than one sensor to get an accurate result. Now, the number of possible navigation tasks for different types of sensors would be big enough to be impossible to be configured at the same time on the FPGA plane. Because of that, the use of ROS is the right choice for autonomous cars that need a high-performance environment in which the time is very important.

Table 5.9 presents some of possible autonomous car activities and the tasks needed to achieve that functionality based on number of research works such as [165, 166]. Moreover, it shows the sensors used to achieve that functionality. Some of these

actions can be achieved by using more than one method, so the system can decide which method has to be used based on the available slots or other criteria. R4THOS in this case is the decision maker based on different processing aspects. As shown in the table, the traffic sign recognition and detection is done using two tasks: detection and recognition. The function is a bit more complex and can be achieved through executing the tasks sequentially. In [167], the research shows that this function has to use the following tasks: detection (hue calculation and detection and morphological filtering) and recognition (labeling, candidate scaling, and template matching). All these tasks have to be configured on the FPGA as tasks through R4THOS on the available hardware tasks slots.

Table 5.9 Autonomous Car Possible Tasks

Function	Tasks	Sensors
Car parking	✓ object recognition ✓ space measurement ✓ object distance	✓ Camera ✓ Radar (detect objects) ✓ sensor (transmit and receive)
Traffic Signs recognition	✓ Sign detection ✓ Sign recognition	✓ Camera
Traffic light detection and reading	✓ Sign detection ✓ colour detection	✓ Camera
Measuring Objects speed	✓ object distance ✓ object detection	✓ sensor (transmit and receive) ✓ Radar (detect objects)

R4THOS is the heart of the autonomous car, which controls all activities and receives all task outputs. These outputs are then processed to be converted to other activities in a continuous cycle. Note that tasks have different sizes and shapes. .

5.5 Summary

Although they can process data very efficiently, a wide range of available commercial cameras is implemented on ASICs, which cannot provide flexibility to the implemented application. Moreover, these cameras are exposed to obsolescence

over time, as they cannot be upgraded because of the nature of the ASIC environment.

This chapter presented an implementation of the DPR imaging system that provides significant flexibility to imaging applications. In addition, the implementation reduces the utilisation area to enable implementing complex applications on small FPGAs. Moreover, this implementation shows the reduction in power consumption compared to the static equivalent designs. An enhanced version of the implementation has been demonstrated in which two reconfigurable regions have been added to increase the system flexibility based on task size and to mask the configuration time through a task pre-fetching feature. The power consumption, resource utilisation, and execution time readings have been measured and reported.

To get the most out of the implemented DPR imaging system, a proposal of the system that is integrated with R4THOS for autonomous cars has been discussed. This system automates the car navigation on the road by swapping tasks in and out through the ICAP primitive on the FPGA. The task swapping is controlled by the components of R4THOS, which works to execute tasks on time using the scheduler, allocator, and configuration manager.

Chapter 6 : Reliable Dynamic Partial Reconfiguration Imaging System

6.1 Introduction

The SRAM-based configuration memory used in FPGAs is highly sensitive to radiation. This radiation can cause corruption in the data stored in the configuration memory in the form of SEUs or MBUs, which can lead to functionality faults in any part of the implemented system. With the increase in the density of configuration cells in FPGAs, the probability of faults increases. However, the new SRAM technologies have better upset reliability but are still susceptible to upsets at a lower rate. According to a study carried out in a Xilinx white paper [61], the 7-series has a total rate of 100 FIT/Mb for all elements compared to 260 FIT/Mb in Virtex-4 (1FIT/MB = 1 upset per10^9 hrs per 10^6 bits) (See Figure 6.1).

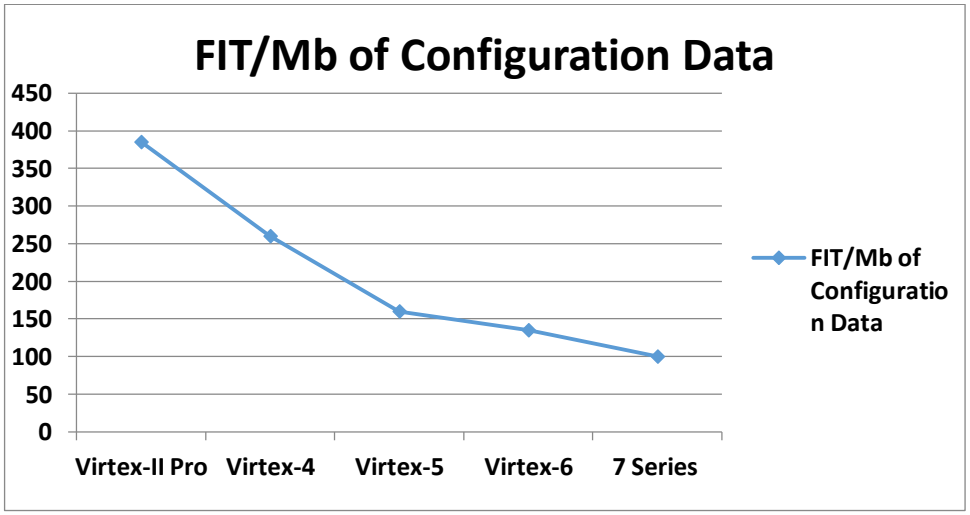


Figure 6.1 Xilinx FPGA Failure rate by product [61]

The DPR imaging environment presented in the previous chapter is a datapath-based environment where the elements are connected in a pipelined manner to form a datapath from the input side to the output side. To enable the implemented DPR imaging system to operate effectively in a harsh environment such as space or

radiation areas, the overall fault tolerance of the FPGA should be enhanced based on the elements of the environment. Many techniques have been proposed to enhance the fault tolerance of FPGAs. Some of these techniques rely totally on the internal configuration port in FPGAs for online access to the configuration memory for detection and correction mechanisms, and some of them use it only for correction, as the detection is done using a different method.

This chapter addresses the reliability issues of the imaging system proposed in chapter 5 and proposes a novel ways to improve it using the available techniques for both the reconfigurable part and the static part. The solutions are mainly based on 7-series FPGAs. The novel contributions of this chapter can be summarised in few points: first, the SEM IP ([63] and [64]) is integrated into the system to evaluate the system overall reliability by injecting errors and applying a correction mechanism. Second, a Built-in Self Test (BIST) using CRC is proposed to enhance the reliability of the system and reduce the recovery time and system downtime with low resources overhead. Third, a TMR is applied to the most important part of the system, which is the RR to increase the system reliability and reduce the resource overhead. These techniques are evaluated in their own or in combination to be adapted to the target applications.

6.2 Related Work and Xilinx FPGAs reliability features

6.2.1 Related Works

Previous works only discuss the reliability issues in FPGAs in general and ways to overcome them. Some of these works use DPR and others do not. In [168], an internal readback scrubber in a Xilinx Virtex-4 FPGA is demonstrated. Moreover, a TMR of the scrubber is used to increase the reliability of the ICAP access point. The work exploits the ECC parity bits embedded in the configuration memory frames to detect and correct single-bit errors in each frame. The work can detect but not correct MBUs in the system, which may leads to system failure if they accumulate. Other systems use a stored golden configuration in an external memory module. This golden bitstream can be configured if a fault is detected or periodically configured

without any fault detection to overwrite any possible faults in the system, as discussed in [65]. This technique can correct any number of errors except the errors in memory elements such as BRAMs, FF, or the LUTs that are configured as distributed RAM or SRL because they are dynamically changed. These elements must be masked during the scrubbing process to eliminate any chance of corrupting the syndrome bits because of the dynamic values of the elements. In [169], a TMR design was implemented to detect errors in the output side of the system. The disadvantage of TMR is its area overhead, as the design should be triplicated. In [170], only the most critical part of the system uses TMR, while the remaining parts do not. This work uses DPR to partition the system into parts to reduce the area overhead. In addition, permanent fault mitigation is discussed in many works. The techniques in these works are based on avoiding damaged resources by loading a new configuration at runtime or relocating the module to a new location on the FPGA. This new location should contain the exact resource layout. One of these techniques is the BIST solution, which loads a specialised test circuit in a specific region at runtime to test the internal logic. Combining more than one technique can increase the reliability of the system, as discussed in [171], according to the data criticality, the expected SEU rate, and the operating window as shown in the mitigation scheme matrix shown in Figure 6.2.

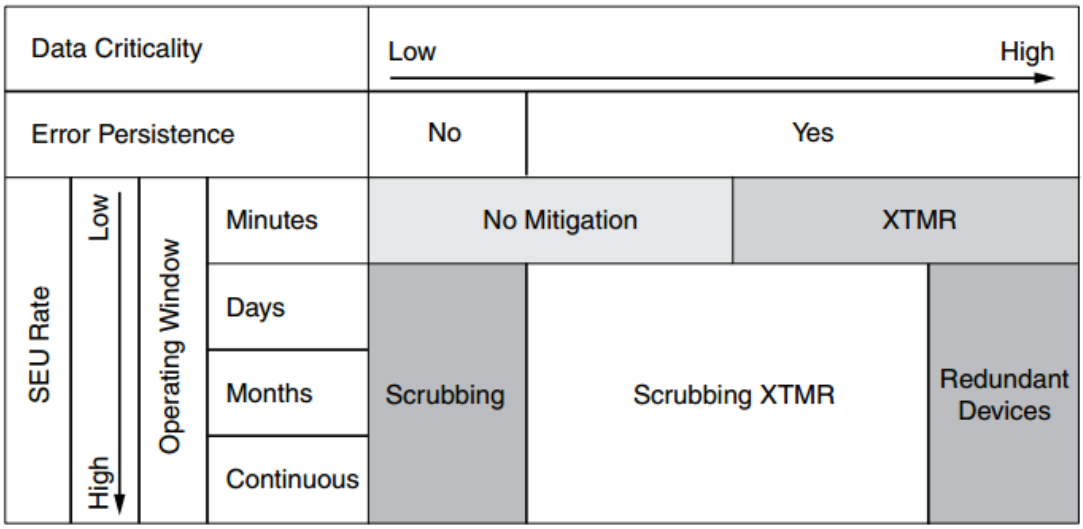


Figure 6.2 Mitigation Scheme Matrix [171]

6.2.2 Reliability Features in Xilinx FPGAs

In Xilinx FPGAs, each configuration frame contains a number of ECC bits that can be used to detect possible bit-flips in the configuration memory. These parity bits can be used in ECC-readback scrubbing schemes to detect errors within each configuration frame. The number of ECC bits in configuration frames differs in each FPGA family. In 7-series FPGAs, there is a 13-bit SECDED (Hamming code) parity value located in the middle word of each configuration frame -the 51st- (with the exception of BRAM frames). These parity bits allow for correcting a maximum of a single-bit error within a frame. Double-bit errors in configuration frames can also be detected but cannot be corrected. To address this limitation, Xilinx has introduced the SEM IP core for their newer FPGA families as an internal scrubber that allows for automatic detection/correction of faults in the configuration memory. The next section discusses the SEM IP and its integration in the imaging system in detail.

In addition, CRC is used in FPGA as a verification parameter at the time of configuration. At the time of the bitstream generation, the BitGen software tool generates a CRC value for every bitstream and then is written to the CRC register. This redefined CRC code is compared to the calculated code using the CRC logic in the board to detect any fault in the configuration process. Moreover, it is used as detection and correction code for the entire FPGA configuration or for individual frames. The CRC code can correct more than one bit, unlike the ECC code.

However, the use of these embedded bits in readback scrubbing schemes cannot be efficient all the time in every case. These schemes are unable to detect faults in FPGA resources that are configured as memory elements, such BRAMs, FF, or LUTs that are configured as distributed RAM or SRL because their values are dynamically changed. These blocks are either automatically bypassed or masked from the readback data when performing the readback operation. Xilinx FPGAs contain a dedicated ECC logic circuit, which is activated at the time of a readback operation. The ECC logic generates a 13-bit syndrome value for each frame, which indicates the location of the flipped bit if any exist. Table 6.1 shows the possible syndrome values and the meaning of each syndrome output. The flipped-bit can be in

the frame data or inside the syndrome data. In some cases, the syndrome value could be a useless value, as it cannot indicate what the expected error situation is.

Table 6.1 ECC Syndrome Codes for Virtex-6 and 7 series

Syndrome (12)	Syndrome (11:0)	Meaning	Detection	Correction
(12) = 0	(11:0) = 0	No bit errors	-	-
(12) = 1	(11:0) != 0	One bit error	Yes	Yes
(12) = 0	(11:0)!= 0	Two bit errors	Yes	No
Unpredictable syndrome		More than two bits	No	No
(12) = 1	(11:0) = 0	Parity bit error	Yes	No

6.3 Integration of SEM IP in the Imaging System

Xilinx has implemented the SEM IP to be integrated in systems as an internal scrubber, which allows for automatic detection and correction of configuration memory faults. It has different levels of error correction capabilities. The first method uses the ECC values to correct one bit and detect up to two bits in each frame, as described in previous related works. The second method can correct double-bit errors through an ECC and CRC combination algorithm. A golden CRC value for each configuration frames is calculated after configuration by performing an initial readback operation. The CRC values are generated by passing the configuration frames into a dedicated CRC generator. These CRC values are stored in BRAM blocks to be used in later readback operations with generated ECC parity bits to correct up to double-bit in each frame. The third method uses a golden bitstream copy to replace the configuration memories in need or periodically using external memory.

Although the SEM IP allows for improving the readback scrubbing capabilities, it has two main limitations. The first limitation is that the IP cannot be used alongside

partial reconfigurable systems, which needs to load different RMs at runtime. Each time a RM is configured; the CRC values should be recalculated and replace the old CRC values. This feature is not supported yet in the IP. The second limitation of the IP is that the scrubber cannot test a specific region in the FPGA like the ICAP scrubber proposed in [46] to test RMs, which leads to reduced scrubbing time.

Moreover, the SEM IP core supports error injection at any place in the FPGA to help in evaluating the system reliability. In this work, the injection feature is used to determine the functional error rates for different RMs and the entire system. Since the SEM IP has no support for the DPR system, an IP software reset is performed after each partial reconfiguration to trigger the IP to calculate new CRC values and therefore release the CRC error flag. The work assumes that the IP supports DPR systems and has no software reset delay at each partial reconfiguration attempt. In addition, the SEM IP core and the imaging system are connected to the ICAP primitive at the same time. In this work, a multiplexing mechanism is proposed to share the ICAP primitive over time between the scrubbing core and the imaging system, which needs to access the ICAP at the time a new module is configured. The following presents the implementation, the fault injection controller, and the experimental results and analysis.

6.3.1 Implementation

Not all the features of the SEM IP are needed in this work. Because of that, many changes have been made to the SEM IP interface to be compatible with the imaging system and its need to use the ICAP primitive (See Figure 6.3). The monitor part in SEM interface has been removed to eliminate any delay when sending a command to the IP as mentioned in the IP manual. The implementation of the modified SEM IP interface alongside a DPR imaging system is depicted in Figure 6.4 and Figure 6.5. All the activities of the SEM IP are controlled by two ports: inject strobe and inject address. The inject strobe is the enable signal that confirms the commands sent by the inject address port. According to the SEM IP manual [63], the state of the IP can be changed by sending commands to the aforementioned ports. There are two main states for the SEM IP core: observation and idle states. In the observation state, the

IP observes the FPGA configuration memories for indications of error conditions. In the idle state, the IP stays idle and waits for error injection action or moves to observation again. In this implementation, the data sent to the IP is divided into two parts based on their meanings. An LSB address is dedicated for the address of the bit within a frame; note that each frame consists of 101 32-bit words therefore a total of 12 bits are needed to determine the location of the bit within the 101 words. The MSBs address is dedicated for the address of the frame within the FPGA based on the physical addressing scheme. The most 5 bits of MSB are used to determine the next state and the enable signal. The physical addressing scheme has fixed number of bits for each address sub-category based on the FPGA family.

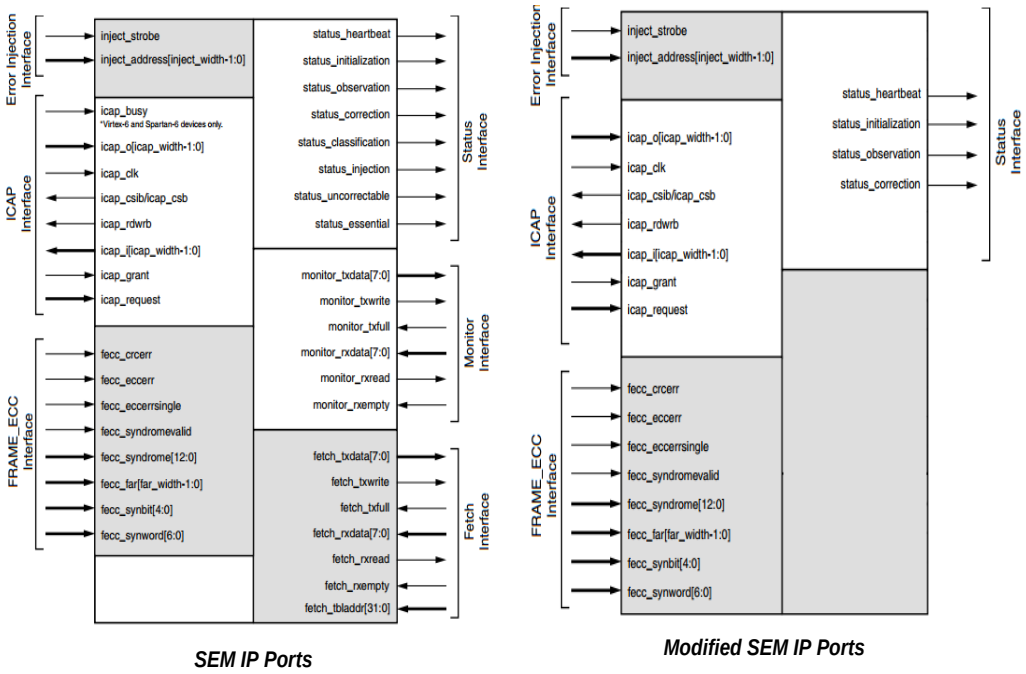


Figure 6.3 SEM IP controller ports

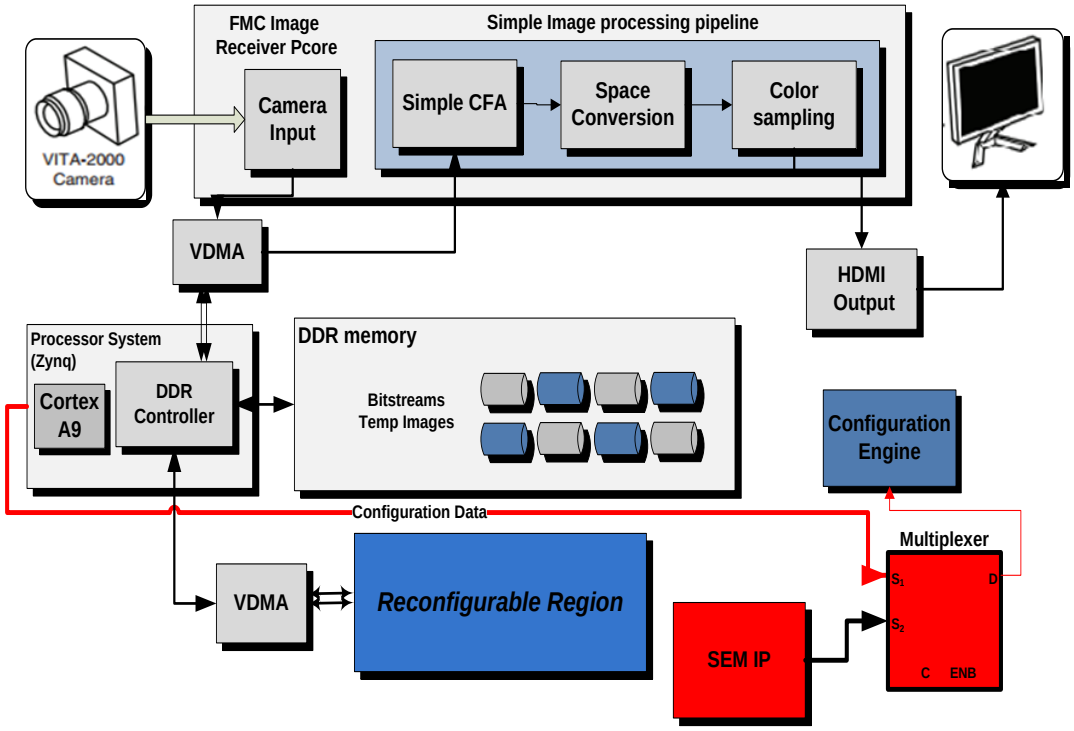


Figure 6.4 The Imaging System Scrubber-based Implementation

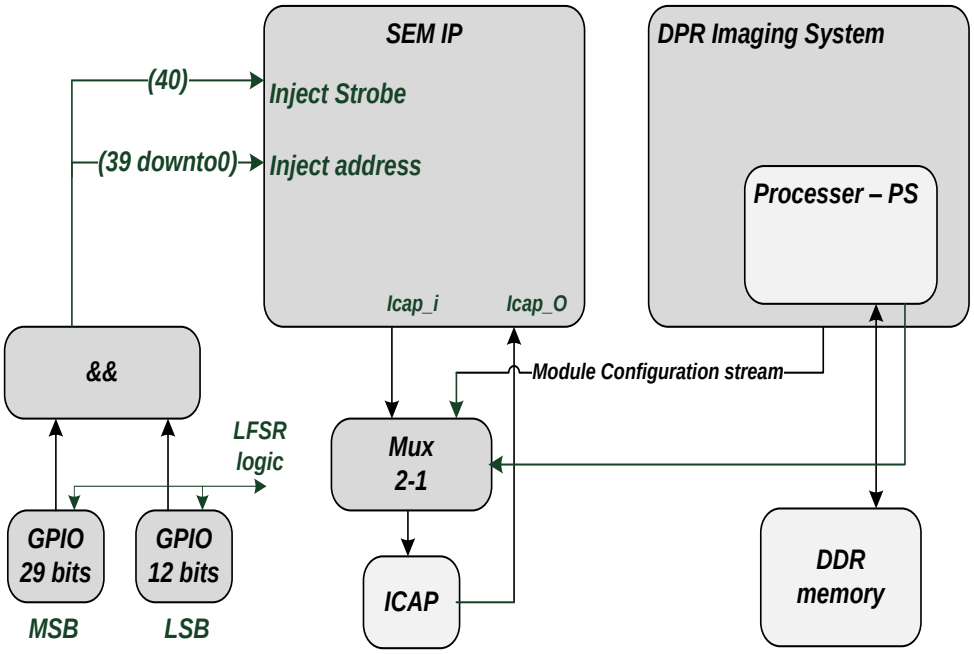


Figure 6.5 The Imaging System Scrubber-based Implementation - details

The following commands show the way of moving between states using the 41 bits controlled by the processor side in our system.

- Move to idle: MSB: 1Exx xxxx, LSB: xxx
- Move to observation: MSB: 1Axx xxxx, LSB: xxx
- Software Reset: MSB: 1Bxx xxxx, LSB: xxx
- Injection(while idle): MSB: 1xxx xxxx, LSB: xxx

Where x is the physical address of the frame and the selected bit.

In addition, the implementation multiplexes the ICAP configuration data over time between the scrubber IP and the imaging system. The imaging system needs to use the ICAP only at the time a new module is configured; otherwise, the SEM IP uses the ICAP for configuration memory error injection and observation activities. When the ICAP primitive is utilised by the imaging system for configuring new tasks, the system cannot detect errors using the core because there is no a direct connection with the internal configuration memory. Figure 6.6 shows the multiplexing logic.

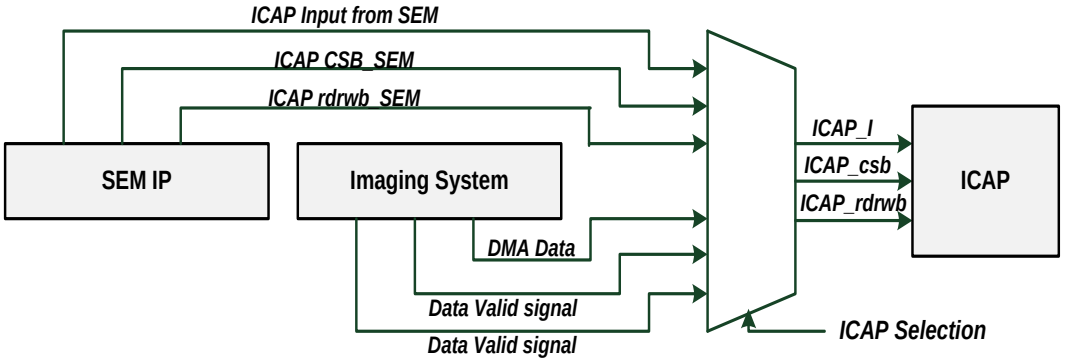


Figure 6.6 ICAP data Multiplexing

6.3.2 Fault Injection Controller

The reliability of the system is evaluated by testing the system operations after injecting errors randomly in the configuration memory. In this work, the errors have been injected in all parts of the system, but for testing purposes, the recorded results have been gathered from the errors injected in the RMs. Three modules are configured one after the other in the same reconfigurable region. To generate random addresses within the boundaries of the reconfigurable region, information extracted

from the generated bitstream are used to indicate the start address of the region, and the total number of configuration words represents the region configuration memory. The physical address representation follows Figure 6.7, where B is the bit address, W is the word address, M is the frame address, C is the column address, R is the row address, H is the half address, and T is the block type. Based on the start address and the location of RM and its shape, the number of columns and rows of the module can be extracted. Figure 6.8 shows the location of RM within the FPGA.

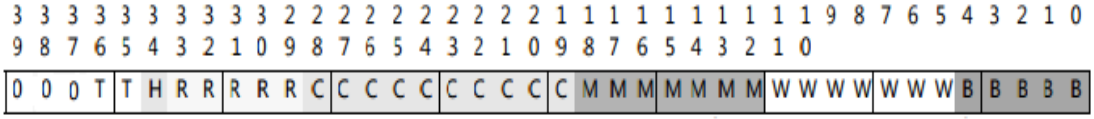


Figure 6.7 Physical address

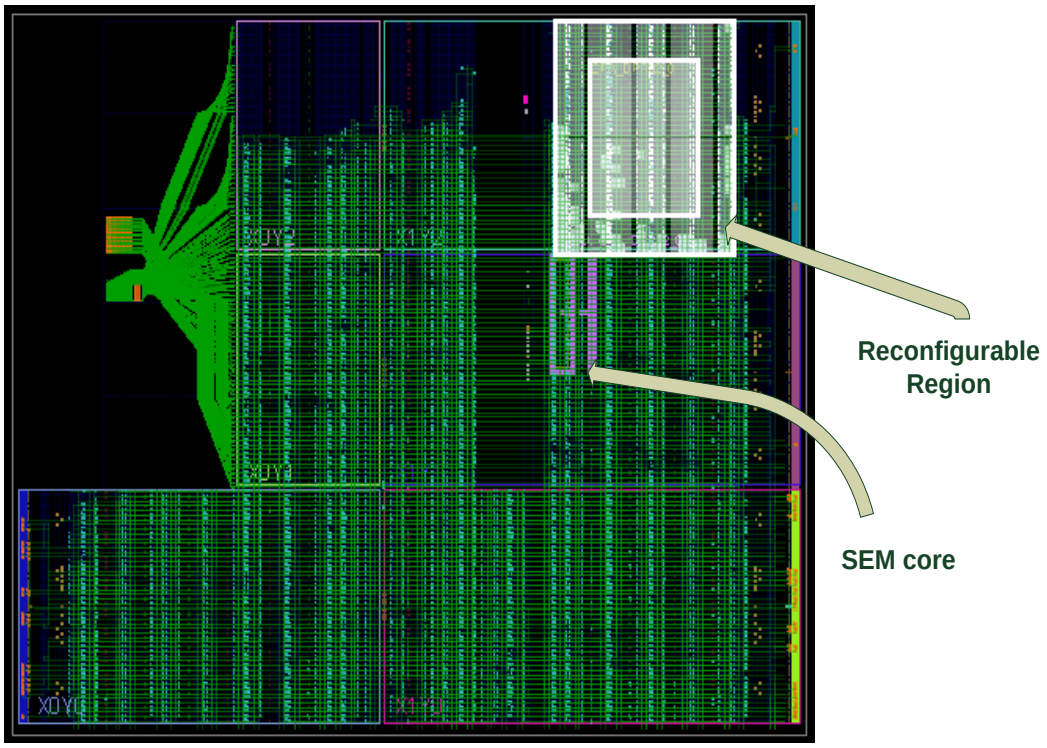


Figure 6.8 location of RM within FPGA

Based on the testing platform and the used 7-series FPGA family, the address of the injected bit should be within the range of the available frames in each column, and words in each frame. According to the 7-series FPGA family, the CLB column has

36 frames. This number of frames in each column can be changed based on the resource type at that specific column. For example, a BRAM column has 128 frames in column high. Linear Feedback Shift Register (LFSR) logic is used to generate a random number for each part of the physical address as represented in Figure 6.8. Note that the maximum address for each part is less than the maximum number by the allocated bits; therefore, modulus logic is used to generate addresses within the boundaries as shown in Figure 6.9.

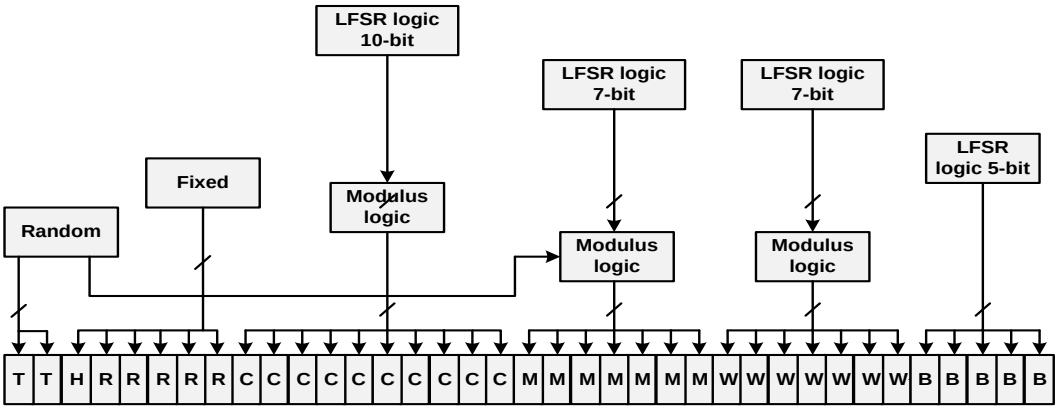


Figure 6.9 LFSR logic

The mechanism of fault behaviour classifications is shown in the data flow presented in Figure 6.10. The mechanism aims to determine how the implemented system performs under configuration memory upsets. Before discussing the mechanism, note that not all upsets can lead to system error, depending on the device utilisation level and the optimisation level of the place and route tool. Therefore, not all the configuration bits are significant. There are three possible behaviours when the errors are injected into the module: (1) nothing happens and the task generates the correct output; (2) the task stops at a specific point and the system waits for a done signal forever; this error can be detected if the module exceeds the expected time for executing the task; or (3) the output data of the task and the stored golden image for the exact task are not matched; this type of error cannot be detected in normal cases.

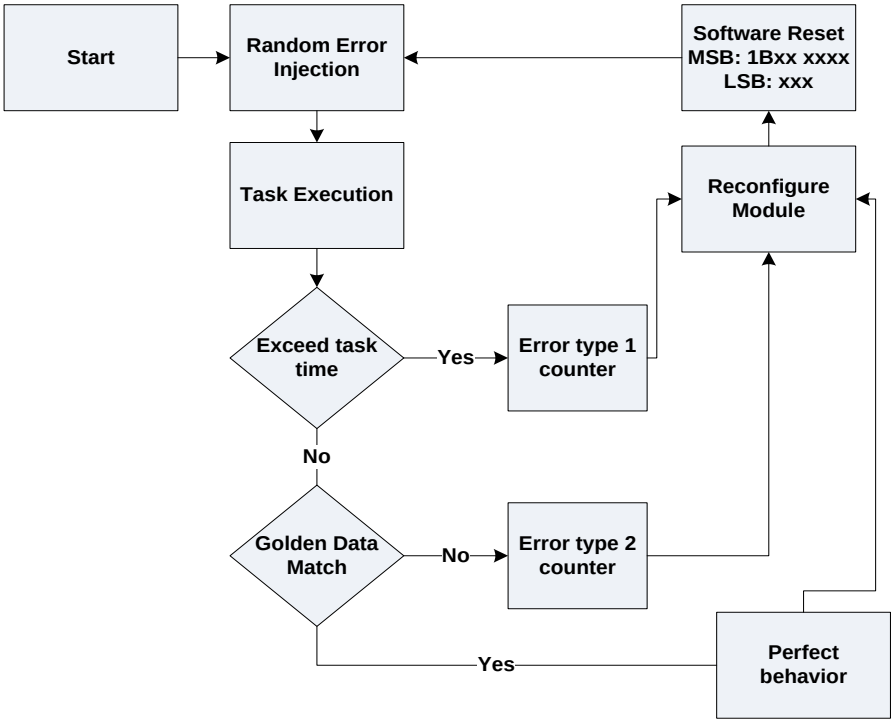


Figure 6.10 Faults injection and system behavior

6.3.3 Fault Injection Results and Analysis

With the aid of the data flow presented in Figure 6.10 and with no fault-tolerant mechanism, the system has been tested with three different modules based on the information provided in the previous chapter to examine the system behaviour under faults. Based on the flow, only one fault is injected in the configuration memory at a time. There are two types of errors. The first type of error can be detected in the form of system hanging where all operation stops at specific point in the task. The second type of errors cannot be detected, as the errors appear in the form of changes in the output data. The scrubber in this section can fix these single upsets and detect double upsets. Table 6.2 shows that an average of 5.62% of the injected errors can lead to a system functionality failure. Around 88% of these failures are in the form of incorrect output data, which cannot be detected. The variation in fault type percentages is due to the types of components used in each module. However, when the scrubber is used, the system can reduce the total numbers of upsets to almost of zero if single frame upsets are injected in non-memory configuration elements. The

percentage of uncorrected upsets can be increased if double errors are injected in the same frame. Note that memory-based configuration elements are masked in the newer FPGA families, so changing its content using the SEM IP core is not possible. Upsets in block RAMs and LUTRAM can cause uncorrected errors which is not supported by the SEM scrubber.

Table 6.2 Faults Types and System Behaviour

Module	***Total Injections***	***Exceed Task Time***	***Golden Image Mismatch***
CFA	105965	535	7490
Percentage		0.5%	7.07%
AWB1	102548	1090	4255
Percentage		1.06%	4.14%
AWB2	98466	576	3546
Percentage		0.5%	3.6%

To show the effect of using the scrubber in the system, the system availability is measured under different ratios between the frequency of scrubbing and the frequency of SEUs. The data provided in Table 6.2, shows one SEU for each scrubbing operation. To make it more general, different scrubbing rates are used to show the system availability for different cases. The availability is the percentage of time that the system is available to operate and produce correct readings. To measure it, the system is considered as unavailable if incorrect system readings are generated. Figure 6.11 shows the system availability for different modules using different scrubbing rates. The rate of scrubbing is varied from 0.01 to 10, which indicates the ratio between the scrubber frequency and SEU rate. When the rate is 0.01, 100 SEUs are injected in each scrubbing operation while the rate of 10 indicates one SEU injection every 10 scrubbing operations. Based on the figure, it is clear that system availability is dropped if the rate goes below one. Because of that, the use of scrubber all the time for error detection and correction is required to increase the system

availability as shown in the figure. In other words, the frequency of scrubbing should be equal or more than SEU rate to increase the system availability.

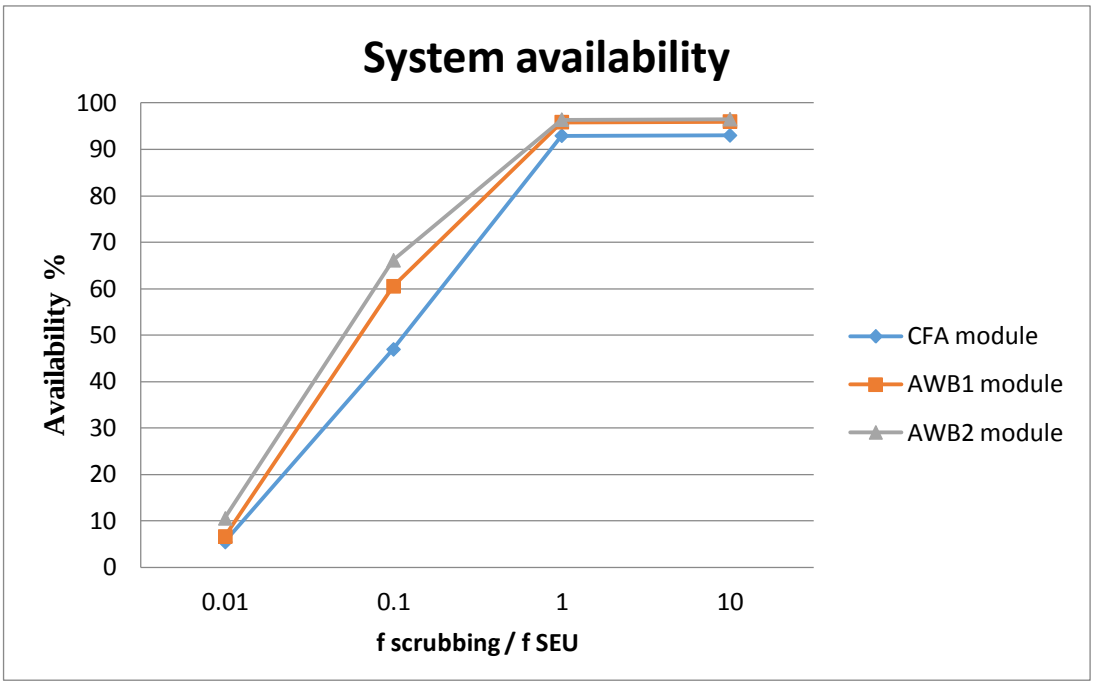


Figure 6.11 Scrubber-based system availability

As mentioned, to get the most out of the scrubber, it should utilise the ICAP primitive all the time for error detection and correction activities. The current work examines the effect of continued use of the ICAP primitive on power consumption. Figure 6.12 shows the power consumption of the logic side of a Zynq-7000 SoC board when the ICAP is utilised by the SEM IP core. The TI Fusion Power Designer tool is used to monitor the power information on the Zynq board. This tool communicates with the embedded power regulators and PMBus controller inside the Zynq board over a serial bus to fetch the power information. The figure shows that the power consumption is not affected by the use of the scrubber and continued use of the ICAP primitive. As shown in the figure, the level of consumed power remains at the same level when the scrubber is active and not active. This observation encourages us to use the scrubber for detecting and correcting errors. However, the power has not been measured when the ICAP primitive is over-clocked, which is not recommended in the normal cases.

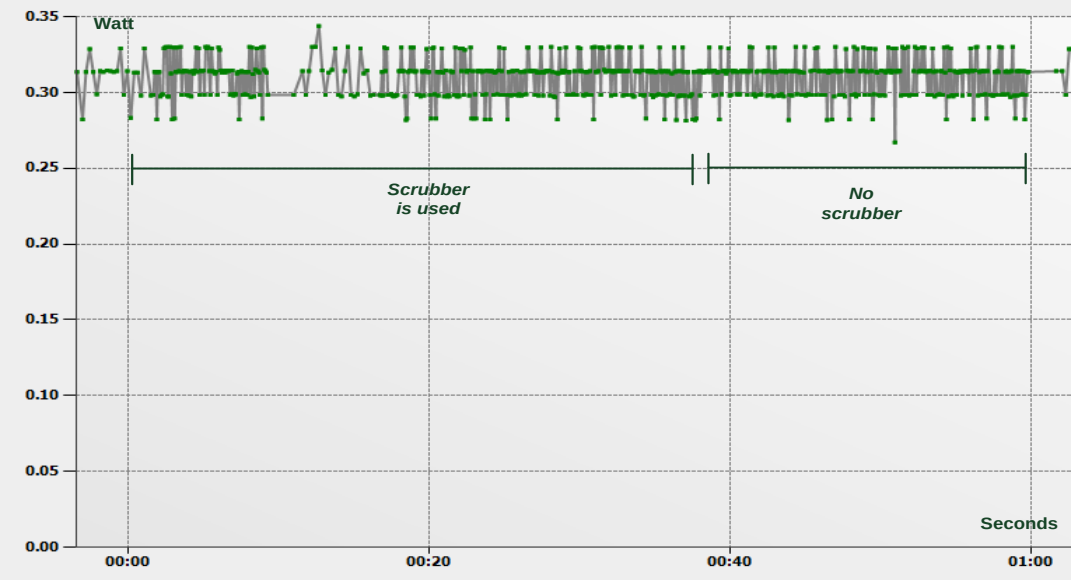


Figure 6.12 Impact of scrubber on power consumption

6.4 Built-In Self Test Solution

The readback scrubber scheme in the previous section cannot detect or correct more than one fault in each frame, which is known as double frame upsets. Moreover, the scrubber cannot deal with the FPGA resources configured as memory elements such as BRAMs, FF, or LUT-based distributed RAM or SRL. These elements represent the program state rather than the program logic. Therefore, these elements need to be masked at the time of mitigation process. Because of the mentioned limitations of that scheme and the high resource overhead of the TMR solution, a CRC-based scheme that uses DPR is implemented in this section to reduce the overall downtime, recovery time, resource overhead for system compared to other methods, and widen the range of possible data correction.

The proposed scheme exploits the datapath nature of the proposed imaging system to divide the design into CRC-based segmented paths. Each path block contains two additional components in addition to the main function: a data-pattern generator and a CRC generator. The data-pattern generator generates suited data for each block that can be processed by the block main task while the CRC generator generates a 32-bit CRC code for the entire output data. The idea is to pass the generated data into the block function and compare the computed CRC output value on the other side with a

pre-generated CRC value kept in the design BRAM. The pre-computed CRC value is generated using the exact same data pattern as the function's input data. The scheme is periodically executed to keep the system fault-free before any real operation. The scheme can be optimised by using only one data-pattern generator at the very beginning block and many CRC generators distributed over the blocks one in each block, whereas the input data comes from the path's previous block. This practice reduces the resource overhead compared to the previous version. Figure 6.13 shows the proposed BIST fault-tolerant design for datapath-based systems. In addition, as the imaging system is a memory-to-memory-based system, the scheme can be used to detect any fault while the data reside in the memory side by placing two CRC components at the input and output sides of the memory to verify the data before passing them to the RM logic, as shown in Figure 6.14. In this method, the CRC value for the data stored in the memory is kept in the BRAM block until the next data fetch. If any changes happened to the data while they are in the DDR memory, the CRC component can flag it. As the RM is regularly changed, the data need to be fetched again in the same path for the next RM. In the same manner, in Figure 6.14, the data that pass through the VDMA logic can be verified by placing CRC generators at the two sides of VDMA and comparing the two values later.

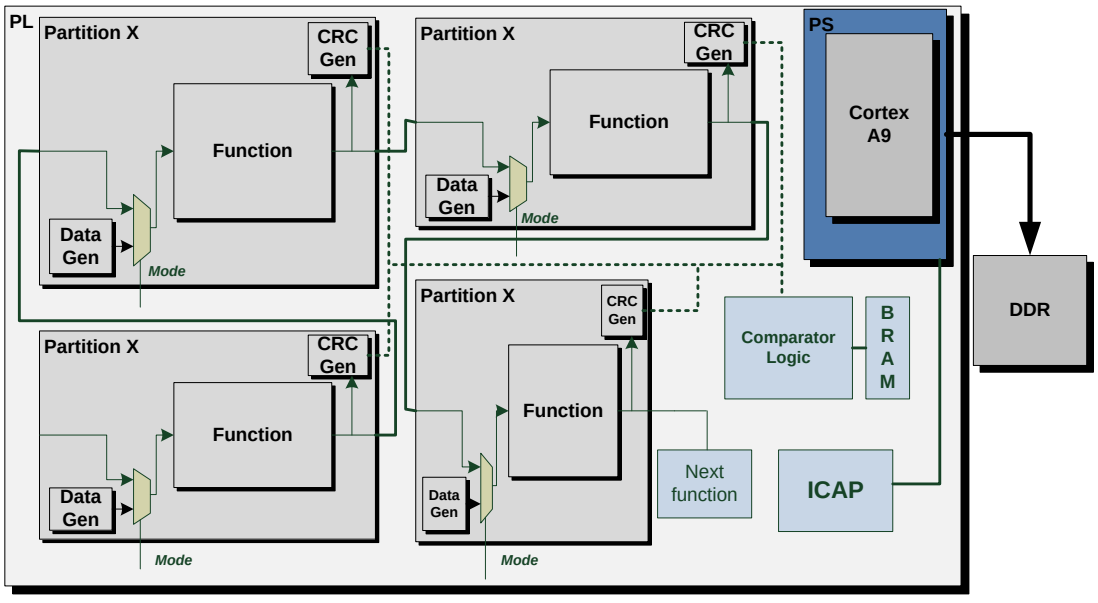


Figure 6.13 BIST appraoch block diagram

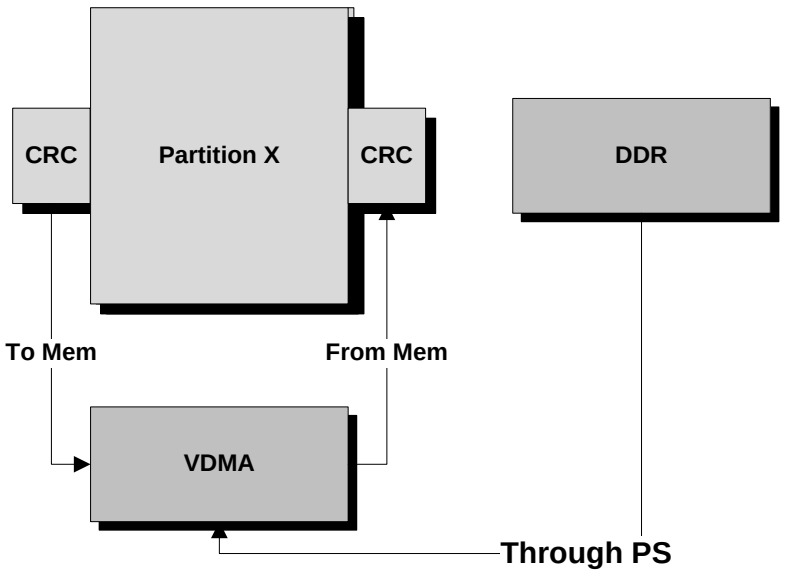


Figure 6.14 Memory-to-memory-based BIST detection mechanism

6.4.1 Implementation

The proposed design in Figure 6.13 contains the CRC and data generators in every partition. To enable the soft error mitigation technique, each block should be implemented as an RM with a fixed RR on the FPGA to make sure that all routes related to that module are included within the RR boundaries. It should follow the DRP design flow explained in chapter 2 to generate the partial bitstream for each RM. As mentioned previously, the RR should contain 20% more resources compared to the largest allocated RM for routing purposes. In addition, to apply DPR to the system, RR should be decoupled from the surrounding components at the time that a new module is configured; otherwise, unknown data will be sent to or from the reconfigurable region during the configuration process. The decoupling process includes resetting the RM and disconnecting all the connected buses using isolating signals, as shown in Figure 6.15, where Res(0:0) prohibits the M00_AXI bus to send and receive data from (CFA_DPR32_0) RM. Moreover, gpio_io(0:0) resets the RM before any configuration attempt. These signals will be released after the configuration process.

The generated CRC values should be compared with the previously generated values to detect any faulty partition. These CRC values are stored in the BRAM at the time of loading the system full bitstream, each with an identifier to link it to a specific partition. All CRC values are stored in the same dual Block RAM for fast access. The CRC values can be retrieved at the next clock cycle. When a mismatch between the stored value and the computed values is detected, the comparator logic triggers the ARM processor to configure the bitstream allocated for that specific partition using the aforementioned configuration steps. All partial bitstreams in this work are stored in the DDR memory but in the real implementation, the bitstreams should be stored in hardened EEPROM or flash to increase the reliability on the memory side. Figure 6.16 shows the floor planning of the implemented BIST scheme on Zynq-7000 AP SoC. It shows two independent data paths in the system. The first datapath belongs to the reconfigurable part of the imaging system where only the RM and the logic memory interface, while the second datapath belongs to the static part where the blocks are connected to each other to form a path from the input side to the output side. The design in Figure 6.15 proves the concept of the scheme; thus, not all the components of the static parts are part of the partitions in the implemented example.

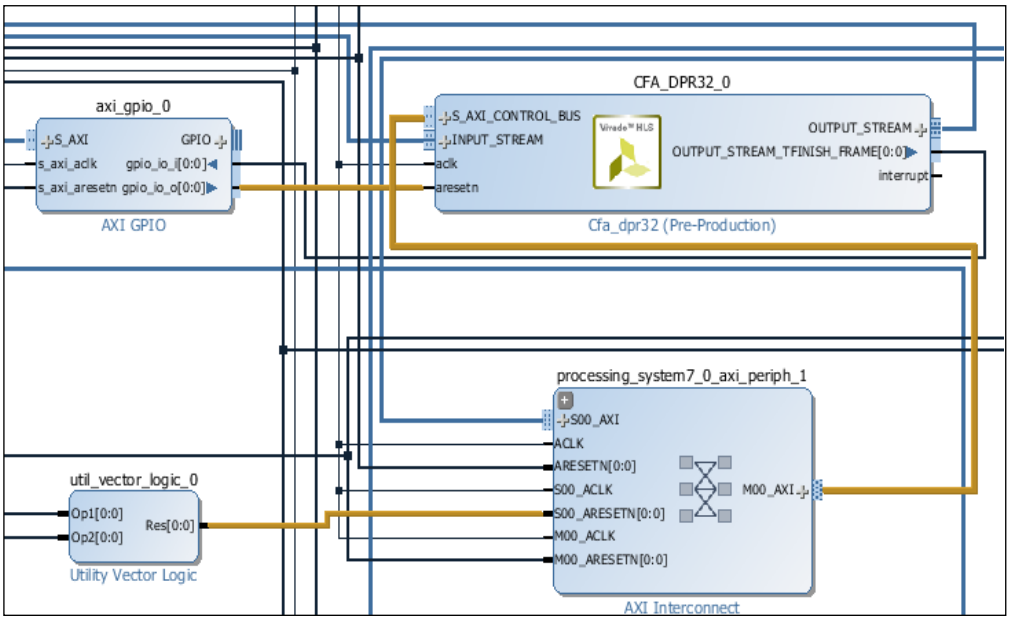


Figure 6.15 Isolation process of the RM

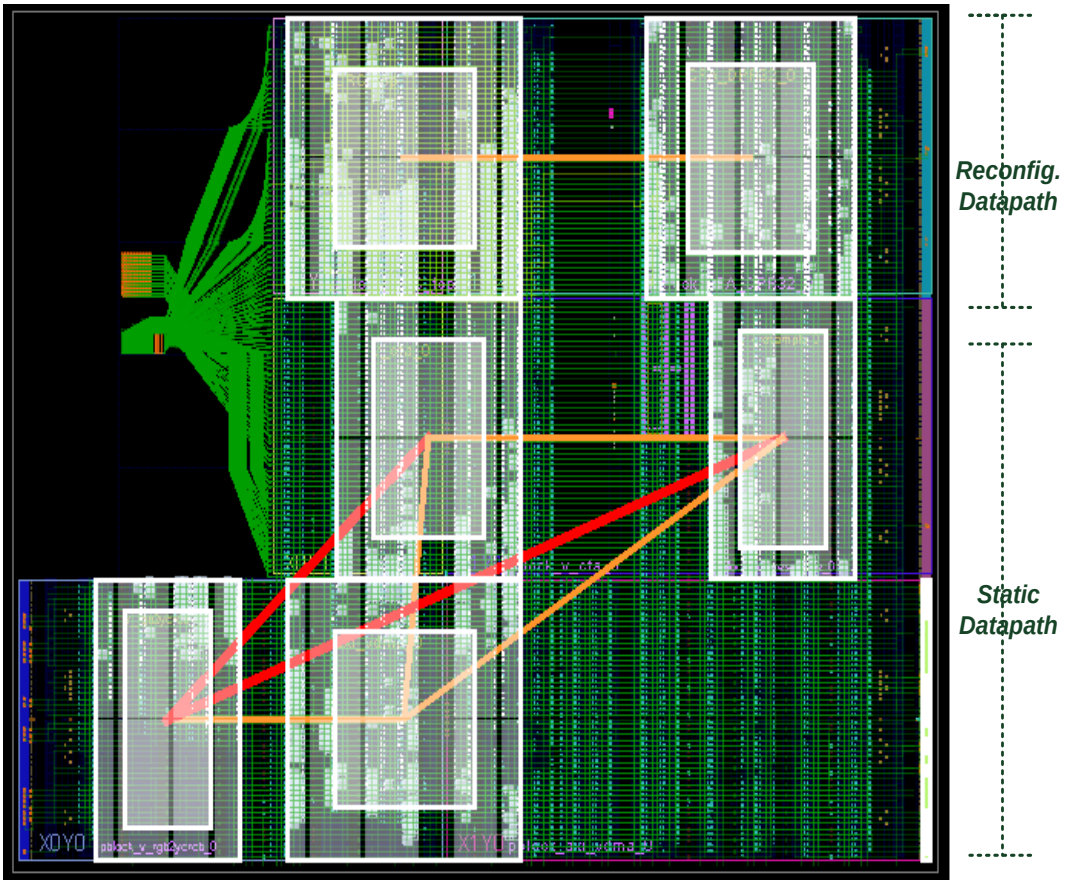


Figure 6.16 BIST scheme floor-planning

6.4.2 Generators and the schemes resource overhead

The Data pattern generator generates a specific pattern that suits the function in the partition in terms of width and length. The generator is connected to the system processor to synchronise its functionality with other components. In contrast, the parallel CRC generator processes a 32-bit word every clock cycle to process the streaming data. The HDL code of the generator has been generator using the Xilinx CRC tool [172]. The CRC generator is based on the CRC32 module polynomial as shown in equation (6.1).

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1 \qquad (6.1)$$

The scheme's resource overhead in each partition is shown in Table 6.3. It shows that each partition should have 20% extra resources because of the use of the DPR

feature. However, the generator utilises a small area footprint compared to the system. Moreover, the scheme has much less resource overhead compared to the full TMR scheme where the system needs to be triplicated to implement the full TMR system.

Table 6.3 BIST Schemes' Resources Overhead

Component	LUT	%	FF	%
Data pattern generator	81	0.15%	67	0.07%
CRC generator	230	0.43%	110	0.1%
Function	Function resources + 20%			

6.4.3 Fault recovery procedure

The scheme was tested with various scenarios, including memory-to-memory and passing through a function scenarios. Memory-to-memory intends to detect any faulty logic in the path to/from the memory and the memory itself, while the other scenario checks faults in function operations. The SEM IP core has been used to inject multiple errors in the same frame. These flips cannot be fixed through the SEM IP itself. The behaviour of the system is checked at the time that errors are injected. The calculated CRC value will be available at the end of the task when all the output data pass through the CRC logic. At that time, the stored CRC value for that particular block is fetched from the BRAM and compared to the calculated one. The ARM processor on the PS side of the Zynq -7000 AP SoC takes care of all these operations. When a fault is detected, the recovery scheme fetches the appropriate bitstream from the DDR memory to be configured through the ICAP after changing the ICAP control from the SEM IP to the recovery scheme. The recovery scheme uses the same configuration engine used for configuring new modules in the imaging system to utilise the full speed of the ICAP primitive. The recovery time depends on the size of RM and the speed of the configuration engine. Table 6.4 and Table 6.5 show the recovery time and the correctable upsets using the proposed scheme for the

two scenarios; for the static and reconfigurable regions. Note that the RRs of the system have different sizes based on the complexity of the function. The detection depends on the complete set of output data. Because of that, each task is performed with the minimum set of input data to reduce the detection time. A set of 30K pixels is used (1 pixel per cycle). As shown in the table, the recovery time is much lower than the whole FPGA recovery time, which is approximately 9.65 ms. Moreover, only that part of the system has to be shut down at the time of recovery, while the other parts are functioning. The mean of functioning here is either doing detection and correction activities because each partition has its own input and output or doing normal activities as the system has two type of datapaths; the static path and memory to memory path. The system can use one of them while the other is recovering. The implemented recovery scheme can correct most of the errors that occur in the region, as the whole region is reconfigured again including the BRAM and DSP components. Some errors cannot be corrected, as they infect clock resources that connect the whole system or static routes that pass through the region. Note that no errors occurred in the memory side. The scheme can periodically test the system for error every 1.4 ms, which is less than the 9.65 required for scrubbing the entire FPGA. Moreover, the system can provide the same level of reliability as the TMR with only small resource overhead.

Table 6.4 Error Correction using BIST Scheme

Module	***Total Injections /Total errors***	***Correctable errors (% of total injections / total errors)***	***Uncorrectable errors (% of total injections)***
To/From Memory path	109170 / 10826	10320 (9.5% / 95.3%)	506 (0.46%)
Reconfigurable function	103566/ 9563	9173 (8.86% / 95.9%)	390 (0.38%)
Static part function	78466/ 6774	6503 (8.3% / 96%)	271 (0.35%)

Table 6.5 Recovery Time for the Proposed Scheme

Module	*RM size (KB)*	*Detection Time (ms) (30K pixels)*	*Configuration time (ms)*	*Total Recovery time (ms)*
To/From Memory path	468	Length of data pattern (0.2)	1.2	1.2 + 0.2 = 1.4
Reconfigurable function	436	Task time ~ (0.2)	1.11	1.11 + 0.2 = 1.31
Static part functions (selected one)	374	Task time ~ (0.2)	0.95	0.95 + 0.2 = 1.15

This scheme exploits the datapath nature of the imaging system to protect all paths using a low amount of resources overhead compared to full TMR approach presented in many works in the literature such as [67]. Moreover, the time of error detection and correction for the scheme is less than the time of normal scrubbing schemes for such FPGA board, which increases the possibility of detecting errors before it accumulate. Finally dividing the system into blocks prevents the error to propagate to next block because all routes are placed within the boundaries of the region.

The above tables are based of injecting two faults in the same frame after each new block reconfiguration. This means only two bit-flips are placed on the block at a time. To investigate how periodically the BIST scheme should be executed to get the most out of the scheme, the availability of the system is measured with many injection rates. This shows how the accumulated bit-flips can lead to system failure at the end. Figure 6.17 shows the availability of the system under different upset rates and different scheme executing rates. Y axis show the system availability percentage while the X axis shows the ratio between the times of performing the recovery scheme and the upset rate. The figure allows us to investigate the best mean time between scheme executions to avoid the accumulation of soft errors in the system. It is clear from the figure that the scheme should be executed with a rate

equal or greater to the SEU rate. The SEU rate depends on the environment and other factors.

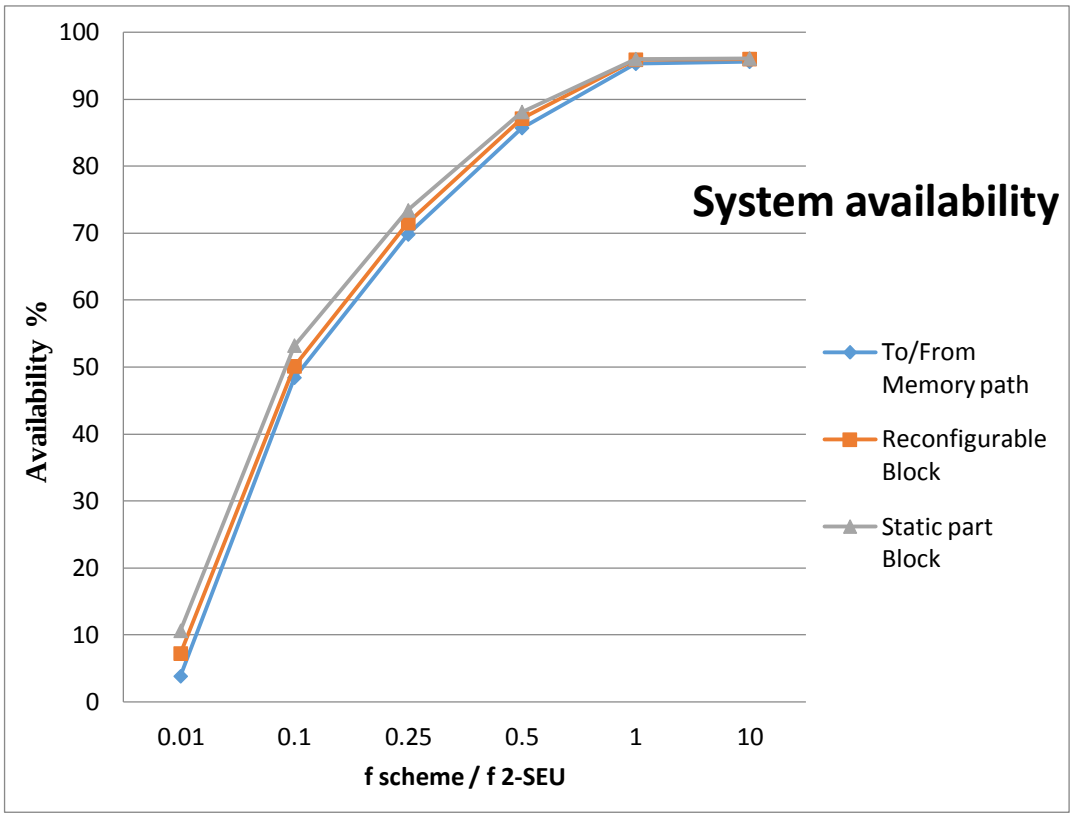


Figure 6.17 BIST-based system availability

6.5 TMR Scheme

Unlike the previous schemes, TMR needs more logic, as the core must be triplicated to vote between many outputs to find the correct output online using the majority method. In the case of limited resource environments, it is hard to apply TMR to the complete design logic. In this section, TMR is applied only to the most important part of the system to decrease resource overhead. Based on the structure of the imaging system discussed the previous chapter, the most important part is the reconfigurable part, where tasks are swapped in and out over time. Figure 6.18 shows the proposed TMR scheme, which is applied to the imaging system.

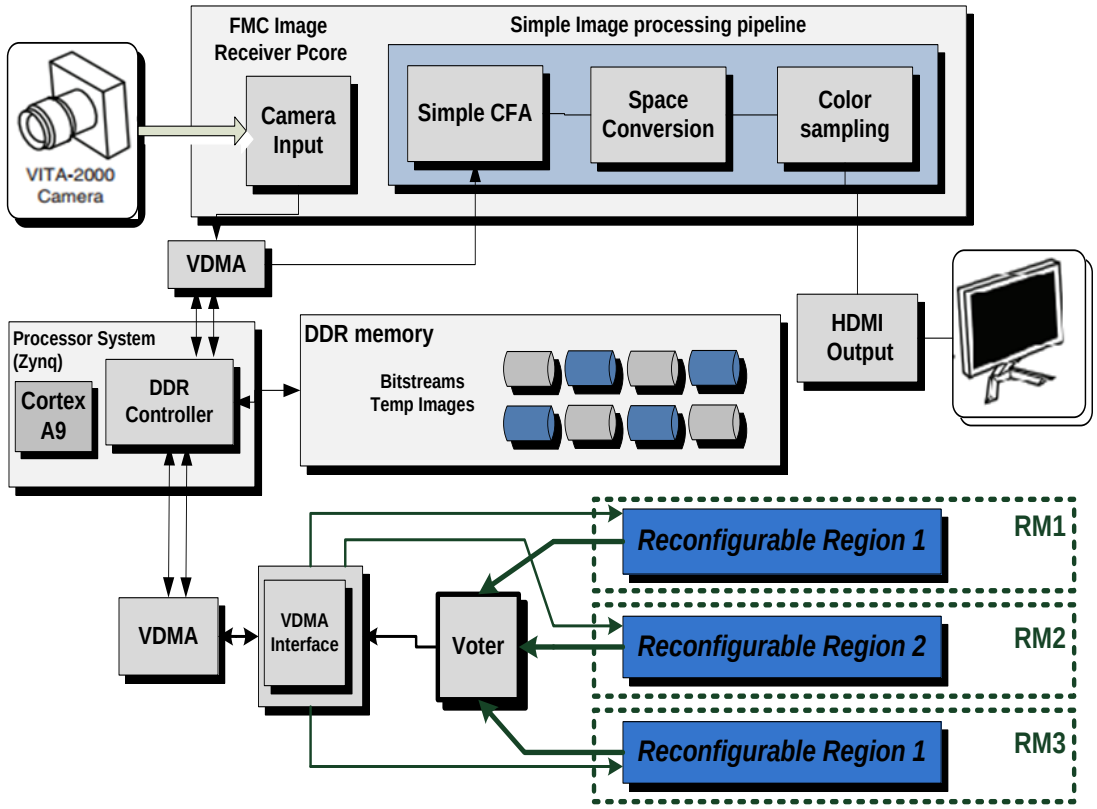


Figure 6.18 The proposed TMR scheme

In the proposed imaging system, the reconfigurable part is connected to VDMA core, which is a one-to-one connection that has a handshaking process to connect the reconfigurable part with the DDR memory over the memory controller in the PS side of the Zynq-7000 board. In the proposed TMR scheme, it is not possible to connect one VDMA to three reconfigurable regions, so a VDMA interface is implemented to triplicate the data and to receive the output from the voter; otherwise, three VDMA cores should be used in the system, which leads to extra resource overhead. There are two criteria in designing the TMR scheme for the imaging system. The first is to reduce the possibility of single faults affecting more than one redundant module. The second is the ability to recover a specific module that is affected by faults. Although the TMR is applied to the reconfigurable part of the system, fulfillment of the previous criteria in TMR must follow the same means of implementing a single reconfigurable part. Each part of the TMR should be implemented in a different RR. In normal TMR deployments, the faults that affect more than one redundant module

are caused by a change in the routing that is shared between the two modules. This could be a problem when placing three redundant logic modules in the same area. Implementing each redundant module in a different RR is the solution for such problem because the DPR tool forces the local routes of these modules to be constrained within the boundaries of the RR. Moreover, at the time of implementing RR-based TMR, a unique partial bitstream is generated for each RR. These partial bitstreams are used to recover an individual RR in the case of a faulty region. The recovery by a partial bitstream includes all types of soft errors in the configuration memory. When a fault is detected, the recovery process takes place by fetching and configuring the partial bitstream from DDR memory using an ARM processor alongside the configuration engine. At this time, the TMR scheme is converted to a DMR, where only two redundant modules are used to detect any possible faults at the time of recovering the third module. The proposed TMR scheme differs from the CRC-based scheme in section 6.4 as this scheme is an online scheme, while the CRC scheme works prior to the real operations of the imaging system to test system readiness.

Because of applying TMR to the reconfigurable part of the system, each time a new task is configured, the system controller needs to configure three copies of the module in RRs. This operation needs 3X the time compared to a single partition, which reduces the system performance. To reduce the configuration time, an MFW feature in Xilinx FPGAs is used as part of the configuration engine, as shown in the work proposed in [164]. The MFW feature allows the writing of multiple configuration frames containing the same content once instead of configuring them individually. To use this feature, the redundant modules of the TMR should be constrained in places that have the same shape and same resource layout and order. This way of implementation faces a problem when relatively big reconfigurable regions are used in small FPGAs where it is not easy to find three identical regions to fit the three big modules. The implemented configuration engine that supports MFW feature follows the flow chart shown in Figure 6.19 to configure multiple bitstream that have the same content. The speed up in this method over the normal method

depends on the size of the relocatable module and the number of modules, which are three modules in our case.

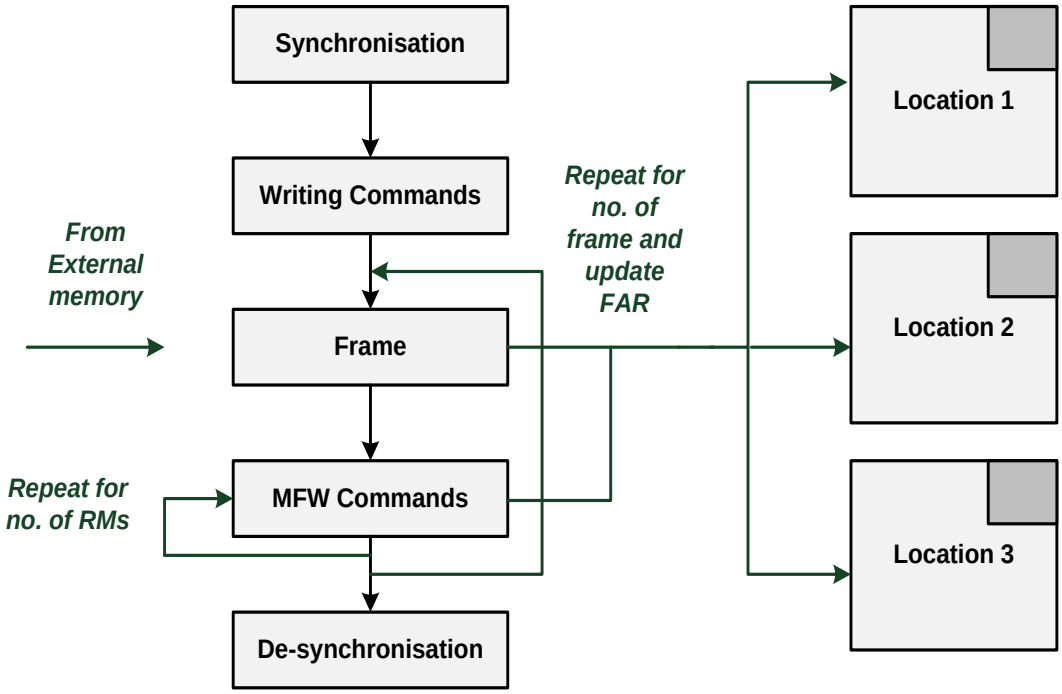


Figure 6.19 Multiple Task configurations using MFW [164]

- **Combining schemes**

Each scheme of the previous proposed schemes has single points of failure in its structure, e.g., the CRC logic in the CRC-based schemes and the voter logic in the TMR. A combination of the three schemes can reduce the overall number of single points of failure. In the case of a single fault in the CRC or voter logic, the scrubber can correct faults. However, partitioning the design into blocks including the SEM IP core makes it easy to recover the SEM logic in the case of any fault with less recovery time. Moreover, the system with combination schemes can reduce faults in the idle state using the scrubber and BIST solution or, on its running state, using the scrubber and the TMR-based solutions. Based on Figures 6.11 and 6.17, the use of combined scheme can increase the overall system availability further more by addressing the limitation of each scheme in the other available one.

6.6 Summary

The SRAM-based configuration memory used in FPGAs is highly sensitive to radiation in form of soft upsets. However, the new SRAM technologies have better-upset immunisation but are still susceptible to upsets at a lower rate. Therefore, implementing FPGA-based systems that work in high radiation environments needs to adopt special techniques to enhance the overall fault tolerance of the FPGA, which lead to lower upset levels.

This chapter has presented a reliable version of the implemented imaging system presented in chapter 5. The work adopts many techniques to enable the system to work in high radiation environments or in the space. The Xilinx SEM IP core was used to evaluate the system reliability under upsets by injecting errors at various blocks and locations in the FPGA configuration memory. Moreover, the core has been used as scrubber to correct single upsets in non-memory-based configuration memory. In addition, due to the data path nature of the system, a novel BIST technique has been implemented to detect and correct any number of errors in the specified region using data and CRC generators at each block. The technique shows very good efficiency in handling errors using small resource overhead. Finally, TMR technique has been used with most important part of the system to reduce the total resource overhead. Moreover, MFW feature has been used to speed up the configuration speed for the identical task used in the TMR technique.

Chapter 7 : Conclusion and Future Work

To address the limitations in the current available imaging systems, this thesis presents a reliable, reconfigurable, flexible imaging system, which aims to increase the overall flexibility of the developed imaging platform compared to the available state-of-art imaging platforms. This thesis investigates the development of a customised imaging platform, which can build various image-processing applications. In this context, the following aspects have been discussed while implementing the system: flexibility, high-performance, power consumption, and reliability. The implemented imaging system significantly exploits the DPR feature to enhance the aforementioned aspects compared to static equivalent designs. DPR allows for efficient use of resources by allowing tasks to share the same logic in a time-multiplexed fashion. This will enhance many aspects in the design, including power dissipation and resource utilisation. Moreover, the ability to change or swap tasks allows for developing more flexible applications in a small footprint by changing functions based on the application requirements. In addition, DPR can enhance the reliability of the system by integrating fault detection and recovery capabilities to meet the standards of fault-tolerant applications that operate in harsh environments.

The reminder of this chapter summarises the research work covered in the thesis, and concludes and evaluates the achievements of the thesis and their impacts on the field. Finally, it discusses and highlights the key areas that require improvements alongside the possible future work related to the work covered in the thesis.

7.1 Summary and Concluding Remarks

The current start of art imaging systems are focusing on performance, power dissipation and ignoring other aspects. To address these limitations in the hidden

areas on the available imaging systems, this thesis presents a reliable, reconfigurable, flexible imaging system, which aims to increase the overall flexibility of the developed imaging platform compared to the available state-of-art imaging platforms. The work in this thesis enables the development of a customised imaging platform, which can be used to build various image-processing applications with no limitation and with high-speed environment.

The main contributions of this thesis are presented in chapters 3, 4, 5, and 6. Chapter 3 proposed a design and architecture of an efficient implementation of the Adams-Hamilton demosaicing algorithm for IPP colour interpolation stage. The work focused on achieving high-performance acceleration of IPP stages using FPGA-based reconfigurable accelerators. The architecture exploited the FPGA parallelism feature to increase the performance by increasing the number of simultaneous executing elements. The experimental results of the implementations showed excellent throughput, lower execution time, reduced power dissipation, and similar image quality compared to the current available solutions. The architecture was implemented using two different approaches: the RTL-based approach and the HLS-based approach. The chapter presented a comparison and evaluation study between the two approaches. Based on the experimental results, the study showed that the RTL-based approach is better in the case when the time and effort are not critical aspects compared to the need of a highly optimised design in terms of area, power, and performance. In contrast, the HLS-based approach is a very good choice for limited time projects. The HLS-based approach can still provide high-performance designs with a slightly higher area utilisation.

Another implementation of the IPP stage is presented in chapter 4 but with a different approach. The chapter presented the design and architecture of an efficient implementation IPP AWB stage. In addition to the FPGA parallelism, the architectures exploited the DPR technology to enhance all design aspects. The novel implementation exploited the nature of the algorithm, which needs to process the image data sequentially using two different set of equations. This means that extra logic is placed on the FPGA without being used. The chapter presented a solution for

these issues by presenting a DPR architecture, which allows for efficient use of resources over time. In addition to the DPR implementation, a static implantation of the algorithms is presented. A comparison between the experimental results of both approaches is performed to show the effect of using DPR on different aspects. The comparison showed that the DPR implementation performs better than the static in all three aspects. DPR implementations showed approximately 40% reduction in power, 30% increase in performance, and 35% lower resource utilisation. An enhanced configuration engine is presented in the chapter to utilise the full speed of the ICAP primitive.

Chapter 5 presented an alternative architecture to ASIC-based commercial cameras because of the lack of thier flexibility. This chapter presented an implementation of a DPR imaging system that provides significant flexibility to imaging applications. In addition, the implementation reduces the utilisation area to enable incorporation of complex applications on small reconfigurable footprint. Moreover, this implementation showed a reduction in power consumption compared to the static equivalent designs. The experimental results showed a good performance, power dissipation, and resource utilisation. An enhanced version of the implementation is demonstrated in the chapter in which two reconfigurable regions have been added to increase the system flexibility to allow a greater range of tasks to be executed according to their size. Moreover, the enhanced version masks the configuration time through a task pre-fetching feature. To obtain the most out of the implemented DPR imaging system, a proposal that integrates the system with R4THOS for autonomous cars is discussed. This system automates the car navigation on the road by swapping tasks in and out through the ICAP primitive on the FPGA. The task swapping is controlled by the components of R4THOS, which works to execute tasks on time using the scheduler, allocator, and configuration manager.

Chapter 6 presented a reliable version of the implemented imaging system by adopting special techniques to enhance the overall fault tolerance and allow the system to operate in harsh environments and to make them resilient to emerging soft upsets.The Xilinx SEM IP core was used to evaluate the system reliability under

different upsets rates. Moreover, the core has been used as a scrubber to correct single upsets and detect two upsets in each frame of the non-memory-based configuration memory frames. The experimental results of the scrubber-based design showed a significant reliability improvement over the non-scrubber-based one. This work presented a novel BIST technique by exploiting the data path nature of the system as the data are passed from the input side to the output side through a specific path. The technique divided the imaging system into different blocks, with each block representing a function. The technique can detect and correct any number of errors in a specific region using data and CRC generators at each block. It showed excellent efficiency in handling errors using small resource overhead. Finally, the TMR technique has been used with the most important part of the system to reduce the total resource overhead. Additionally, the MFW feature has been used to speed up the configuration speed for the identical task used in the TMR technique.

7.2 Future Work

Several aspects of the work presented in this thesis can be further investigated or improved. Some of these are listed below:

- **Task development environment**: In terms of implementing imaging tasks, the HLS-based approach showed how the method compares with the RTL-based approach in terms of performance, development time, and resource utilisation. However, the development time should be further enhanced to allow for efficient task development. This can be achieved by developing a library that contains many imaging components such as window buffers, line buffers, median value calculator, and others, which have a direct relation with imaging algorithms. The library should be embedded in an environment that allows for connecting these components to form a complete algorithm. This way of developing imaging tasks can enhance the overall development time, which allows implementing any number of tasks with less time. Moreover, it enhances the upgrading process and keeps the system up-to-date.
- **Injecting faults**: The current method of injecting faults into the FPGA configuration memory using the configuration port is not enough to ensure the

system reliability. To observe the effect of faults on the reliability of system parts, a real radiation test should be performed, which can represent in a better way the radiation existed in different environments.

- **ICM integration**: SEM IP has been used in this thesis to test the system reliability and to inject faults randomly on the FPGA plane. As mentioned earlier in chapter 6, the SEM core does not support DPR because the CRC value is changed after each reconfiguration operation, which appears as a fault on the FPGA. A workaround is proposed in the chapter, but a better solution is to integrate the ICM proposed in [46, 69, 164]. The ICM is able to replace the SEM core in which the CRC values can be regenerated and embedded in the logic. Moreover, it can work as a scrubber for part of the FPGA, unlike the SEM core. In addition, it supports many configuration operations that enhance the idea of the DPR imaging system.
- **System integration with R4THOS:** The concept of integrating the system with R4THOS for autonomous cars is proposed in chapter 6 but not implemented due to the absence of a complete R4THOS. The idea should be implemented to increase the efficiency of the imaging system and to be integrated efficiently with various numbers of imaging applications.
- **Flexible reconfigurable partition**: The current system has two reconfigurable regions to accommodate imaging tasks to mask the configuration time overhead when swapping tasks in and out. The reason of having two regions is to minimise the total power consumption and total resources while maintaining high flexibility. To increase the system flexibility, a relocation mechanism should be investigated to allow tasks to be reconfigured anywhere within the reconfigurable partition. This method of allocating tasks allows for a greater number of tasks to be allocated in a given region simultaneously. The problem of the general relocation mechanism is summarised in two points. First, current modern FPGAs have a heterogeneous architecture in which components are distributed over the FPGA plane with no existence of uniform areas of identical resources. E.g. the layout of CLB, BRAM, and DSP component in most FPGAs is not uniform in the horizontal axis. This means that it is difficult to place the tasks simply

anywhere in the region because their bitstreams have specific layouts of FPGA resources that allow each task to be reconfigured in limited places within the region. Second, the communication between tasks in the proposed mechanism must be investigated to allow for high-speed connections. Further investigation in this part of the thesis should include different types of FPGAs that have large areas in the FPGA consisting of a uniform arrangement of resource columns.

Reference

References

[1] A. Kaur and M. Kaur, "Review of Image Processing- Introduction to Image Enhancement Techniques using Particle Swarm Optimization," *International Journal of Artificial Intelligence and Applications for Smart Devices,* vol. Vol.3, No.1, pp. 15-20, 2015.

[2] M. G., "Moore's law at 40, Understanding Moore's law: four decades of innovation," pp. 67-84, 2006.

[3] G. McFarland and M. Flynn, " Limits of Scaling MOSFETs," Technical Report CSLTR-95-662, Stanford University, Nov. 1995.

[4] X. Tian and K. Benkrid, "“High-performance quasi monte carlo financial simulation: FPGA vs. GPP vs. GPU," *ACM Transactions on Reconfigurable Technology and Systems,* vol. 3(4), pp. 1–22, 2010.

[5] Xilinx, "Xilinx Ships Industry's First 20-nm All Programmable Devices," *XCELL Journal, Issue 86,* First Quarter 2014.

[6] Xilinx, "Vivado Design Suite User Guide: High-Level Synthesis," *Xilinx Document: UG902,* 2014.

[7] Brattain and Walter, "Bell Labs logbook " in *Bell Labs logbook* ed, pp. 7-8, December 1947.

[8] Wikipedia. (n.d). *Transistor count*. Available: https://en.wikipedia.org/wiki/Transistor_count

[9] S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial," *Design & Test of Computers, IEEE,* vol. 13 , Issue 2, pp. 42-57, 1996.

[10] Xilinx, "Xilinx 15th Anniversary: 15 years of innovation " *XCELL Journal,* Issue 32, Second Quarter 1999.

[11] Xilinx, "Zynq-7000 All Programmable SoC Technichal reference manual," *Xilinx Document: UG585 (v1.10),* 2015.

[12] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation,* vol. 2, pp. 135-253, 2008.

[13] Grand View Research. (2014). *Global FPGA market by application Expected to Reach USD 9,882.5 Million by 2020*. Available: http://www.grandviewresearch.com/press-release/global-fpga-market

[14] D. Manners. (2010). *FPGA Market Soaring To $4bn In 2010, says Gavrielov* Available: http://www.electronicsweekly.com/news/components/programmable-logic-and-asic/fpga-market-soaring-to-4bn-in-2010-says-gavrielov-2010-05/

[15] K. Morris. (2014). *Xilinx vs. Altera Calling the Action in the Greatest Semiconductor Rivalry*. Available: http://www.eejournal.com/archives/articles/20140225-rivalry/

[16] F. Vahid, *Digital Design with RTL Design, Verilog and VHDL*, 2nd edition ed.: John Wiley and Sons, 2010.

[17] S. Singh and D. Greaves, "Kiwi: Synthesis of FPGA Circuits from Parallel Programs," in *16th International Symposium on Field-Programmable Custom Computing Machines, 2008. FCCM '08.* , pp. 3-12, 2008.

[18] W. A. Najjar, W. Bohm, B. A. Draper, J. Hammes, R. Rinker, J. R. Beveridge*, et al.*, "High-level language abstraction for reconfigurable computing," *IEEE Computer,* vol. 36, Issue 8, pp. 63-69, 2003.

[19] Xilinx, "Introduction to FPGA Design with Vivado High-Level Synthesis," *Xilinx Document: UG998,* 2013.

[20] Altera, "Implementing FPGA Design with the OpenCL Standard," *Altera White Paper: WP-01173-3.0,* 2013.

[21] Xilinx, "Vivado Design Suite User Guide: Synthesis," *Xilinx Document: UG901,* 2013.

[22] Xilinx, "Vivado Design Suite User Guide: Implementation," *Xilinx Document: UG904 (v 2015.3),* 2015.

[23] Xilinx, "Zynq-7000 All Programmable SoC Overview," *Xilinx Document: DS190,* 2015.

[24] Xilinx, "7 Series FPGAs Configurable Logic Block " *Xilinx user guide document: UG474 (v1.7),* 2014.

[25] Xilinx, "7 series DSP48E1 slice user guide," *Xilinx Document: UG479 (v1.8),* Novmber 2014.

[26] F. Brosser and E. Milh., "SEU Mitigation Techniques for Advanced Reprogrammable FPGA in Space," *Master thesis, Chalmers University of Technology,* 2014.

[27] Xilinx, "Vivado Design Suite user guide, Partial Reconfiguration," *Xilinx Document: UG909 (v2014.3),* October 2014.

[28] Altera, "Accelerating High-Performance Computing with FPGAs," *Altera White Paper: WP-01029-1.1,* October 2007.

[29] L. semiconductor, "Solving Today's Interface Challenges with Ultra-Low-Density FGPA Bridging Solutions," *A Lattice Semiconductor White Paper,* August 2013.

[30] A. B. Nejad, M. E. Martinez, and K. Goossens, "An FPGA bridge preserving traffic quality of service for on-chip network-based systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-6, 2011.

[31] Xilinx, "High Performance Computing Using FPGAs," *Xilinx White Paper: WP375,* 2010.

[32] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGABased Accelerator Design for Deep Convolutional Neural Networks," in *in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 161-170, 2015.

[33] A. Mitra, M. R. Vieira, P. Bakalov, V. J. Tsotras, and W. A. Najjar, "Boosting XML Filtering Through a Scalable FPGA-Based Architecture," in *In CIDR'09*, Asilomar, CA, USA, 2009.

[34] C. Pham-Quoc, Z. Al-Ars, and K. Bertels, "Heterogeneous hardware accelerator architecture for streaming image processing," in *International Conference on Advanced Technologies for Communications (ATC)* pp. 374-379, 2013.

[35] J. K. Toft and A. Nannarelli, "Energy efficient FPGA based hardware accelerators for financial applications," in *NORCHIP, 2014*, Tampere, pp. 1-6, 2014.

[36] Y. ZHANG, F. ZHANG, Z. JIN, and J. D. BAKOS, "An fpga-based accelerator for frequent itemset mining," *ACM Trans. Reconfigurable Technol. Syst.,* vol. vol. 6, no. 1, pp. 2:1–2:17, May 2013.

[37] W. Wang and X. Huang, "An FPGA co-processor for adaptive lane departure warning system," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, Beijing, pp. 1380-1383, 2013.

[38] D. Koch and J. Tørresen, "Advances and Trends in Dynamic Partial Run-Time Reconfiguration," *Dagstuhl Seminar Proceedings 10281-Dynamically Reconfigurable Architectures,* pp. 1-9, 2010.

[39] P. S. Ostler, M. J. Wirthlin, and J. E. Jensen, "FPGA Bootstrapping on PCIe Using Partial Reconfiguration," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011* pp. 380-385, 2011.

[40] M. Hubner, J. Meyer, O. Sander, L. Braun, J. Becker, J. Noguera*, et al.*, "Fast Sequential FPGA Startup Based on Partial and Dynamic Reconfiguration," in *2010 IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* , pp. 190-194, 2010.

[41] Xilinx, "LogicCORE IP XPS HWICAP v5.00," *Xilinx Document: DS586,* July 2010.

[42] Xilinx, "LogicCORE IP AXI HWICAP v3.00," *Xilinx Document: PG134,* December 2013.

[43] L. Ming, W. Kuehn, L. Zhonghai, and A. Jantsch, "Run-time Partial Reconfiguration speed investigation and architectural design space exploration," in *International Conference on Field Programmable Logic and Applications. FPL* pp. 498-502, 2009.

[44] S. Liu, R. N. Pittman, and A. Forin, "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller," *Microsoft Research, Tech. Rep. MSR-TR-2009- 150,* September 2009.

[45] K. Vipin and S. A. Fahmy, "A high speed open source controller for FPGA Partial Reconfiguration," in *International Conference on Field-Programmable Technology (FPT)*, pp. 61-66, 2012.

[46] A. Ebrahim, K. Benkrid, X. Iturbe, and H. Chuan, "A novel high-performance fault-tolerant ICAP controller," in *Conference on Adaptive Hardware and Systems (AHS), NASA/ESA*, pp. 259-263, 2012.

[47] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," *Embedded Systems Letters, IEEE,* vol. 6, pp. 41-44, 2014.

[48] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "Metawire: Using FPGA configuration circuitry to emulate a Network-on-Chip," in *International Conference on Field Programmable Logic and Applications,FPL*, pp. 257-262, 2008.

[49] R. Bonamy, P. Hung-Manh, S. Pillement, and D. Chillet, "UPaRC--Ultra-fast power-aware reconfiguration controller," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1373-1378, 2012.

[50] L. Ming, W. Kuehn, L. Zhonghai, and A. Jantsch, "Reducing FPGA Reconfiguration Time Overhead using Virtual Configurations," in *In Proceeding of the International Workshop on Reconfigurable Communication Centric System-on-Chips*, Karlsruhe, Germany, 2010.

[51] Altera, "Reducing Power Consumption and Increasing Bandwidth on 28-nm FPGAs," *Altera White Paper: WP-01148-2.0,* 2012.

[52] N. Grover and M.K.Soni, "Reduction of Power Consumption in FPGAs - An Overview," *International Journal of Information Engineering and Electronic Business (IJIEEB),* vol. 4, no. 5, pp. 50-69, 2012.

[53] X. Iturbe, K. Benkrid, R. Torrego, A. Ebrahim, and T. Arslan, "Online clock routing in Xilinx FPGAs for high-performance and reliability," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 85-91, 2012.

[54] M. Hentati, A. Nafkha, P. Leray, J. Fran, C. Nezan, and M. Abid, "Software Defined Radio Equipment: What's the Best Design Approach to Reduce Power Consumption and Increase Reconfigurability ?," *International Journal of Computer Applications (IJCA),* vol. 45, no. 14, pp. 26-32, 2012.

[55] H. Ziade, R. Ayoubi, and R. Velazco, "A Survey on Fault Injection Techniques," *The International Arab Journal of Information Technology (IAJIT),* vol. 1, no. 2, pp. 171-186, 2004.

[56] S. Srinivasan, P. Mangalagiri, X. Yuan, N. Viiaykrishnan, and K. Sarpatwari, "FLAW: FPGA lifetime awareness," in *Design Automation Conference, 2006 43rd ACM/IEEE*, pp. 630-635, 2006.

[57] C. Guerin, V. Huard, and A. Bravaix, "The Energy-Driven Hot-Carrier Degradation Modes of nMOSFETs," *IEEE Transactions on Device and Materials Reliability* vol. 7 , Issue 2, pp. 225-235, 2007.

[58] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *Micro, IEEE,* vol. 23, Issue 4, pp. 14-19, 2003.

[59] Altera, "Introduction to Single-Event Upsets," *Altera White Paper: WP-01206-1.0,* Sep. 2013.

[60] A. A. Gaffar, J. A. Clarke, and G. A. Constantinides, "Modeling of glitch effects in FPGA based arithmetic circuits," in *IEEE International Conference on Field Programmable Technology, FPT* pp. 349-352, 2006.

[61] Xilinx, "7 Series FPGAs Mitigation Single-Event Upsets," *Xilinx White Paper: WP395 (v1.1),* May 2015.

[62] Xilinx, "Virtex-6 FPGA Configuration User Guide," *Xilinx Document: UG360 (v3.8),* August 2014.

[63] Xilinx, "Soft Error Mitigation Controller V4.1- LogiCORE IP Product Guide," *Xilinx Documnet: PG036,* Nov. 2014.

[64] E. Crabill and P. Chang, "Scan-Based Soft Error Mitigation of Configuration Memory in Xilinx 7-Series FPGA Devices," *IP solutions,* Xilinx.

[65] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel*, et al.*, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," *IEEE Transactions on Nuclear Science,* vol. 55, Issue 4, pp. 2259-2266, 2008.

[66] U. Legat, A. Biasizzo, and F. Novak, "SEU Recovery Mechanism for SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science,* vol. 59, Issue 5, pp. 2562-2571, 2012.

[67] C. Carmichael, "Triple-Module Redundancy Design Techniques for Virtex FPGAs," *Xilinx Application Note: XAPP197 (v1.0.1),* July 2006.

[68] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, and A. Astarloa, "A novel SEU, MBU and SHE handling strategy for Xilinx Virtex-4 FPGAs," in *International Conference on Field Programmable Logic and Applications. FPL*, pp. 569-573, 2009.

[69] A. Ebrahim, T. Arslan, and X. Iturbe, "On enhancing the reliability of internal configuration controllers in FPGAs," in *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*, pp. 83-88, 2014.

[70] M. Sarfraz, *Computer Vision and Image Processing in Intelligent Systems and Multimedia Technologies*, 1st ed.: IGI Global, Hershey, PA, USA, 2014.

[71] F. M. Siddiqui, "Deliverable 5: IPPro Optimisations. Report on architectural IPPro optimisations to accelerate data-intensive image processing applications," Deliverable Report, Rathlin Project, August 2015.

[72] J. Zhou, "Getting the Most Out of Your Image-Processing Pipeline," *Texas Instruments white paper,* October 2007.

[73] R. Ramanath, W. E. Snyder, Y. Yoo, and M. S. Drew, "Color image processing pipeline," *Signal Processing Magazine, IEEE,* vol. 22, pp. 34-43, 2005.

[74] B. E. Bayer, "Color imaging array," U.S. Patent No. 3,971,065, July 1976.

[75] Sony. (July 2003). *Sony Press Release*. Available: http://www.sony.net/SonyInfo/News/Press/200307/03-029E/

[76] LogicBricks, "logiISP Image Signal Processing (ISP) Pipeline," *LogicBricks Documents: Data Sheet (v 1.3),* July 2015.

[77] K. Wen-Chung, W. Sheng-Hong, C. Lien-Yang, and L. Sheng-Yuan, "Design considerations of color image processing pipeline for digital cameras," *IEEE Transactions on Consumer Electronics* vol. 52, Issue 4, pp. 1144-1152, 2006.

[78] S.Battiato, A.R.Bruna, G.Messina, and G.Puglisi, *Image Processing for Embedded Devices*: Applied Digital Imaging ebook series 2010.

[79] J. A. Parker, R. V. Kenyon, and D. Troxel, "Comparison of Interpolating Methods for Image Resampling," *IEEE Transactions on Medical Imaging,* vol. 2, Issue 1, pp. 31-39, 1983.

[80] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. 29, Issue 6, pp. 1153-1160, 1981.

[81] W. T. Freeman, "Median filter for reconstruction missing color samples," U.S. Patent No. 4,724,395, 1988.

[82] J. F. Hamilton and J. E. Adams, "Adaptive Color Plane Interpolation in Single Sensor Color Electronic Camera," U. S. Patent, No. 5,629,734, 1997.

[83] T. T. Nguyen, "System and method for asymmetrically demosaicing raw data images using color discontinuity equalization," U. S. Patent, No . 20020167602 A1, 2002.

[84] K. Hirakawa and P. J. Wolfe, "Spatio-Spectral Color Filter Array Design for Optimal Image Recovery," *Image Processing, IEEE Transactions on,* vol. 17, pp. 1876-1890, 2008.

[85] J. W. Glotzbach, R. W. Schafer, and K. Illgner, "A method of color filter array interpolation with alias cancellation properties," in *Proceedings of International Conference onImage Processing, vol.1,* , pp. 141-144, 2001.

[86] E. Dubois, "Filter Design for Adaptive Frequency-Domain Bayer Demosaicking," in *Image Processing, 2006 IEEE International Conference on*, pp. 2705-2708, 2006.

[87] X. Lia, B. Gunturkb, and L. Zhangc, "Image demosaicing: a systematic survey," *Proceedings of SPIE* Vol. 6822, pp. 68221J-1–68221J-15, 2008.

[88] X. Zhao, "High Efficiency Coarse-Grained Customised Dynamically Reconfigurable Architecture for Digital Image Processing and Compression Technologies," PhD Thesis, University of Edinburgh, UK, November 2011.

[89] G. Zapryanov, D. Ivanova, and I. Nikolova, "Automatic white balance algorithms for digital still camera- a Comparative study," *Information Technologies and Control,* vol. 1, pp. 16-22, January 2012.

[90] M. Fedor, "Approaches to color balancing," *PSYCH221/EE362 course project, Department of Psychology, Stanford University, U.S.A,* 1998.

[91] J. Chiang and F. Al-Turkait, "Color balancing experiments with the HP-photo smart-C30 digital camera," *PSYCH221/EE362 course project, Department of Psychology, Stanford University, U.S.A.,* 1999.

[92] D. A. Forsyth, "A novel algorithm for color constancy," *International Journal of Computer Vision,* vol. 5, Issue 1, pp. 5-35, 1990.

[93] A. Gijsenij and T. Gevers, "Color Constancy Using Natural Image Statistics and Scene Semantics," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 33, Issue 4, pp. 687-698, 2011.

[94] Altera, "Video and Image Processing Design Using FPGAs," *Altera White Paper: WP-VIDEO0306-1.1 (v1.1),* March 2007.

[95] D. B. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation," presented at the Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, Monterey, California, USA, pp. 63-72, 2009.

[96] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," presented at the Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays, Monterey, California, USA, pp. 47-56, 2012.

[97] K. Pauwels, M. Tomasi, J. Diaz Alonso, E. Ros, and M. M. Van Hulle, "A Comparison of FPGA and GPU for Real-Time Phase-Based Optical Flow, Stereo, and Local Image Features," *Computers, IEEE Transactions on,* vol. 61, pp. 999-1012, 2012.

[98] C. Bira, L. Gugu, R. Hobincu, V. Codreanu, L. Petrica, and S. Cotofana, "An Energy Effective SIMD Accelerator for Visual Pattern Matching," presented at the in Proc. 4th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, Ediburgh, Scotland, June 2013.

[99] J. Garcia-Lamont, M. Aleman-Arce, and J. Waissman-Vilanova, "A Digital Real Time Image Demosaicking Implementation for High Definition Video Cameras," in *Electronics, Robotics and Automotive Mechanics Conference,. CERMA*, pp. 565-569, 2008.

[100] A. Karloff and R. Muscedere, "A low-cost, real-time, hardware-based image demosaicking algorithm," in *IEEE International Conference on Electro/Information Technology, eit.*, pp. 146-150, 2009.

[101] S. A. Fahmy, "Generalised Parallel Bilinear Interpolation Architecture for Vision Systems," in *International Conference on Reconfigurable Computing and FPGAs. ReConFig '08.* , pp. 331-336, 2008.

[102] X. Tan, S. Lai, B. Wang, M. Zhang, and Z. Xiong, "A simple gray-edge automatic white balance method with FPGA implementation," *Journal of Real-Time Image Processing,* vol. 10, Issue 2, pp. 207-217, 2015.

[103] L. Tian, X. Liu, J. Li, and X. Guo, "Image Preprocessing of CMOS Image Acquisitoon System Based on FPGA," *International Journal of Digital Content Technology & its Applications (JDCTA),* vol. 6, Issue 20, pp. 130-139, 2012.

[104] ASICFPGA. (n.d.). *Camera Image Signal Processing Core*. Available: http://www.asicfpga.com/site_upgrade/asicfpga/isp/isp_core1.html

[105] M. Bergeron, S. Elzinga, G. Szedo, G. Jewett, and T. Hill, "LogiCORE 1080p60 Camera Image processing Reference Design," *Xilinx Application: XAPP794 (v1.3),* December 2013.

[106] P. Greisen, S. Heinzle, M. Gross, and A. P. Burg, "An FPGA-based processing pipeline for high-definition stereo video," *EURASIP Journal on Image and Video Processing,* vol. 2011, pp. 1-13, 2011.

[107] IMPERX, "User Customizable Image Processor," *Specifications sheet* 2013.

[108] C.-H. Chen, S.-Y. Tan, and W.-T. Huang, "A novel hardware-software co-design for automatic white balance," presented at the Proceedings of the 7th Conference on 7th WSEAS International Conference on Multimedia, Internet & Video Technologies - Volume 7, Beijing, China, pp. 203-212, 2007.

[109] F. M. Siddiqui, M. Russell, B. Bardak, R. Woods, and K. Rafferty, "IPPro: FPGA based image processing processor," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Belfast, pp. 1-6, 2014.

[110] P. Yiannacouras, J. G. Steffan, and J. Rose, "Portable, Flexible, and Scalable Soft Vector Processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 20 , Issue 8, pp. 1429-1442, 2012.

[111] G. Hegde and N. Kapre, "Energy-Efficient Acceleration of OpenCV Saliency Computation Using Soft Vector Processors," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM),* , pp. 76-83, 2015.

[112] H. Y. Cheah, F. Brosser, S. A. Fahmy, and D. L. Maskell, "The iDEA DSP Block-Based Soft Processor for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.,* vol. 7, pp. 1-23, 2014.

[113] G. Ansaloni, P. Bonzini, and L. Pozzi, "Heterogeneous coarse-grained processing elements: A template architecture for embedded processing acceleration," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, pp. 542-547, 2009.

[114] C. Brunelli, F. Garzia, and J. Nurmi, "A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities," *Journal of Real-Time Image Processing,* vol. 3, Issue 1-2, pp. 21-32, 2008.

[115] J. C. Chen and C. Shao-Yi, "CRISP: Coarse-Grained Reconfigurable Image Stream Processor for Digital Still Cameras and Camcorders," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 18, Issue 9, pp. 1223-1236, 2008.

[116] S. R. Chalamalasetti, S. Purohit, M. Margala, and W. Vanderbauwhede, "MORA - An Architecture and Programming Model for a Resource Efficient Coarse Grained Reconfigurable Processor," in *NASA/ESA Conference on Adaptive Hardware and Systems, . AHS*, pp. 389-396, 2009.

[117] S. Khawam, I. Nousias, M. Milward, Y. Ying, M. Muir, and T. Arslan, "The Reconfigurable Instruction Cell Array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 16, Issue 1, pp. 75-85, 2008.

[118] Z. Xin, Y. Ying, A. T. Erdogan, and T. Arslan, "Dual-core reconfigurable demosaicing engine for next generation of portable camera systems," in *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pp. 289-294, 2010.

[119] H. Parizi, A. Niktash, N. Bagherzadeh, and F. Kurdahi, "MorphoSys: A Coarse Grain Reconfigurable Architecture for Multimedia Applications," in *Euro-Par 2002 Parallel Processing*. vol. 2400, B. Monien and R. Feldmann, Eds., ed: Springer Berlin Heidelberg, pp. 844-848, 2002.

[120] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *Field Programmable Logic and Application*. vol. 2778, P. Y. K. Cheung and G. Constantinides, Eds., ed: Springer Berlin Heidelberg, pp. 61-70, 2003.

[121] O. Losson, L. Macaire, and Y. Yang, "Comparison of color demosaicing methods," *Advances in Imaging and Electron Physics, Elsevier,* vol. 162, pp. 173–265, 2010.

[122] G. S. Zapryanov and I. N. Nikolova, "Demosaicing Methods for Pseudo-Random Bayer Color Filter Array," in *Annual Workshop on Circuits, Systems and Signal Processing, ProRISC'2005*, Veldhoven, Netherlands pp. 687-692, 2005.

[123] R. Ramanath, W. E. Snyder, and G. L. Bilbro, "Demosaicing Methods for Bayer Colour Arrays," *Journal of Electronic Imaging,* vol. 11, Issue 3, pp. 306-315, 2002.

[124] T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 27 no. 1, pp. 13-18, 1979.

[125] Wikipedia. (n.d). *Median Filter*. Available: https://en.wikipedia.org/wiki/Median_filter

[126] K. Benkrid, D. Crookes, and A. Benkrid, "Towards a General Framework for FPGA Based Image Processing using Hardware Skeletons " *Parallel Computing,* vol. 28 no. 7-8, pp. 1141-1154, 2002.

[127] R. Turney, "Two-Dimensional Linear Filtering," *Xilinx, Xilinx Application Note: XAPP993 (v1.1),* 2007.

[128] G. Hawkes, "Video Scene Coherence, Frame Buffers, and Line Buffers," *Xilinx, Xilinx Application Note: XAPP296 (v1.0),* 2002.

[129] A. R. Rostampour and A. P. Reeves, "2-D Median Filtering and Pseudo Median Filtering," in *the 20th Southeastern Symposium on System Theory*, pp. 554-557, 1988.

[130] Xilinx, "LogiCORE IP Multi-Port Memory Controller (MPMC)," *Xilinx Document: DS643 (v 6.03.a),* 2011.

[131] Xilinx, "Color Filter Array Interpolation v7.0," *Xilinx Document: PG002, V. 7.0,* 2015.

[132] E. Kalali and I. Hamzaoglu, "A low energy 2D adaptive median filter hardware," presented at the Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, pp. 725-729, 2015.

[133] H. NguyenT.K., C. Belleudy, and T. V. Pham, "Performance and Evaluation Sobel Edge Detection on Various Methodologies " *International Journal of Electronics and Electrical Engineering,* vol. 2 No. 1, pp. 15-20, 2014.

[134] M. V. Fernando, C. Kohn, and P. Joshi, "Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool," *Xilinx Application Note: XAPP890 (v1.0)* 2012.

[135] Xilinx, "Virtex-6 FPGA System Monitor," *Xilinx user guide document: UG370 (v 1.2),* 2014.

[136] E. Srikanth. (2014). *Zynq-7000 AP SoC Low Power Techniques part 2 - Measuring ZC702 Power using TI Fusion Power Designer Tech Tip*. Available: http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Low+Power+Techniques+part+2+-+Measuring+ZC702+Power+using+TI+Fusion+Power+Designer+Tech+Tip

[137] Xilinx, "XPower Estimator User Guide," *Xilinx user guide document: UG440 (v 13.4),* 2012.

[138] Xilinx, "Vivado Design Suite User Guide Power Analysis and Optimization," *Xilinx user guide document: UG907 (v 2012.4),* 2013.

[139] Wallpapers wide. (n.d). *1920x1080 HD 16:9 Wallpapers*. Available: http://wallpaperswide.com/1920x1080-wallpapers-r.html

[140] J. Hiraiwa and H. Amano, "An FPGA Implementation of Reconfigurable Real-Time Vision Architecture," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pp. 150-155, 2013.

[141] Xilinx, "9 REASONS WHY THE VIVADO DESIGN SUITE ACCELERATES DESIGN PRODUCTIVITY," *Xilinx Backgrounder,* 2014.

[142] D. Nikitenko, M. Wirth, and K. Trudel, "White-balancing algorithms in colour photograph restoration," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pp. 1037-1042, 2007.

[143] E. Y. Lam, "Combining gray world and retinex theory for automatic white balance in digital photography," in *Consumer Electronics, 2005. (ISCE 2005). Proceedings of the Ninth International Symposium on*, pp. 134-139, 2005.

[144] Xilinx, "LogiCORE IP MultiImage statistics," *Xilinx document: DS752 (v 2.0),* March 2011.

[145] S. Lai, X. Tan, Y. Liu, B. Wang, and M. Zhang, "Fast and robust color constancy algorithm based on grey block-differencing hypothesis," *Optical Review,* vol. 20, pp. 341-347, 2013.

[146] S. Marsi and G. Ramponi, "A flexible FPGA implementation for illuminance–reflectance video enhancement," *Journal of Real-Time Image Processing,* vol. 8, pp. 81-93, 2011.

[147] M. C. Hanumantharaju, M. Ravishankar, and D. R. Rameshbabu, "Design of Novel Algorithm and Architecture for Gaussian Based Color Image Enhancement System for Real Time Applications," in *Advances in Computing, Communication, and Control: Third International Conference, ICAC3 2013, Mumbai, India, January 18-19, 2013. Proceedings*, S. Unnikrishnan, S. Surve, and D. Bhoir, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 595-608, 2013.

[148] J. F. Grcar, "Mathematicians of gaussian elimination," *Notices of the AMS, 58(6):782–792,* 2011.

[149] Wikipedia. (n.d). *Cramer's rule*. Available: https://en.wikipedia.org/wiki/Cramer%27s_rule

[150] Altera, "Building an IP Surveillance Camera System with a Low-Cost FPGA," *Altera Corporation White paper: WP-01133-1.1,* May 2012.

[151] J. Chiang and S. Zammattio, "Five ways to build flexibility into industrial applications with FPGAs," *Altera Corporation White paper: WP-01154-2.0,* September 2014.

[152] A. Oetken, S. Wildermann, J. Teich, and D. Koch, "A Busbased SoC Architecture for Flexible Module Placement on Reconfigurable FPGAs," in *in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, Milan, Italy, pp. 234–239, Aug. 2010.

[153] D. Ziener, S. Wildermann, A. Oetken, A. Weichslgartner, and J. Teich, "A Flexible Smart Camera System based on a Partially Reconfigurable Dynamic FPGA-SoC," in *in Proceedings of the Workshop on Computer Vision on Low-Power Reconfigurable Architectures at the FPL 2011*, pp. 29–30, Sept 2011.

[154] Avnet EM. (Jan. 30, 2015). *"HDMI Input/Output FMC Module."* Available: https://www.em.avnet.com/en-us/design/drc/pages/supportanddownloads.aspx?RelatedId=442

[155] Semiconductor Components Industries, "VITA 2000 2.3 Megapixel 92 FPS Global Shutter CMOS Image Sensor," *On Semiconductor Data Sheet, Order Number: NOIV1SN2000A/D, Rev. 5* July 2013.

[156] Xilinx, "7 Series FPGA Configuration User Guide," *Xilinx Document: UG470 (v1.10),* June 2015.

[157] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision – A Run-time Reconfigurable MPSoC Architecture for Future Driver Assistance Systems," *it - Information Technology,* vol. 49, no. 3, pp. 181–187, 2007.

[158] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, I. Martinez*, et al.*, "R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient, and Dependable Computing on FPGAs," *IEEE Trans. Comput.,* vol. 62, pp. 1542-1556, 2013.

[159] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of ACM, vol. 20, no. 1, pp. 46-61,* 1973.

[160] H. Chuan, K. Benkrid, X. Iturbe, A. Ebrahim, and T. Arslan, "Efficient On-Chip Task Scheduler and Allocator for Reconfigurable Operating Systems," *Embedded Systems Letters, IEEE,* vol. 3, pp. 85-88, 2011.

[161] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, T. Arslan, and I. Martinez, "Runtime Scheduling, Allocation, and Execution of Real-Time Hardware Tasks onto Xilinx FPGAs Subject to Fault Occurrence," *Int. J. Reconfig. Comp,* Article ID 90505, 2013.

[162] X. Iturbe, K. Benkrid, T. Arslan, C. Hong, and I. Martinez, "Empty resource compaction algorithms for real-time hardware tasks placement on partially reconfigurable FPGAs subject to fault occurrence," *Proceedings of the International Conference on ReConFigurable Computing and FPGA,* pp. 27 - 34, 2011.

[163] X. Iturbe, K. Benkrid, A. Ebrahim, H. Chuan, T. Arslan, and I. Martinez, "Snake: An Efficient Strategy for the Reuse of Circuitry and Partial Computation Results in High-Performance Reconfigurable Computing," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pp. 182-189, 2011.

[164] A. Ebrahim, K. Benkrid, X. Iturbe, and H. Chuan, "Multiple-clone configuration of relocatable partial bitstreams in Xilinx Virtex FPGAs," in *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*, pp. 178-183, 2013.

[165] R. Yusnita, F. Norbaya, and N. Basharuddin, "Intelligent parking space detection system based on image processing," *The International Journal of Innovation, Management and Technology,* vol. 3, no. 3, pp. 232-235, June 2012. .

[166] W. Yu and T. Chen, "Parking space detection from video by augmenting training dataset," *IEEE International Conference on Image Processing, pp. 849–852,* 2009.

[167] S. Waite and E. Oruklu, "FPGA-Based Traffic Sign Recognition for Advanced Driver Assistance Systems," *Journal of Transportation Technologies, Vol. 3, No. 1, pp. 1-16,* 2013.

[168] J. Heiner, N. Collins, and M. Wirthlin, "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing," in *Aerospace Conference, 2008 IEEE*, pp. 1-10, 2008.

[169] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Design, Automation and Test in Europe, 2005. Proceedings*, Vol. 2, pp. 1290-1295, 2005.

[170] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, pp. 226-232, 2006.

[171] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-Event Upset Mitigation Selection Guide," *Xilinx Application Note, XAPP987 (v1.0), March 2008,* 2008.

[172] Xilinx, "IEEE 802.3 Cyclic Redundancy Check," *Xilinx Application Note: XAPP209,* 2001.